# ขั้นตอนวิธีการจัดทำดัชนีสำหรับข้อความไทยที่มีความผิดพลาด

นาย วรวัฒน์ วรศิลป์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2542

ISBN 974-334-630-9

ลิขสิทธิ์ของ จุฬาลงกรณ์มหาวิทยาลัย

# INDEXING ALGORITHM FOR THAI TEXT WITH ERRORS

Mr. VORAVAT VORASILP

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science in Computer Science
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 1999

ISBN 974-334-630-9

| | |
|---|---|
| หัวข้อวิทยานิพนธ์ | ขั้นตอนวิธีการจัดทำดัชนีสำหรับข้อความไทยที่มีความผิดพลาด |
| โดย | นาย วรวัฒน์ วรศิลป์ |
| ภาควิชา | วิศวกรรมคอมพิวเตอร์ |
| อาจารย์ที่ปรึกษา | ผู้ช่วยศาสตราจารย์ ดร.สมชาย ประสิทธิ์จูตระกูล |

---

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

..... คณบดีคณะวิศวกรรมศาสตร์
(ศาสตราจารย์ ดร.สมศักดิ์ ปัญญาแก้ว )

คณะกรรมการสอบวิทยานิพนธ์

..... ประธานกรรมการ
( อาจารย์ จารุมาตร ปิ่นทอง )

..... อาจารย์ที่ปรึกษา
( ผู้ช่วยศาสตราจารย์ ดร.สมชาย ประสิทธิ์จูตระกูล )

..... กรรมการ
( ผู้ช่วยศาสตราจารย์ เมธี ศรีสังวาล )

..... กรรมการ
( ผู้ช่วยศาสตราจารย์ วิวัฒน์ วัฒนาวุฒิ )

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

วรวัฒน์ วรศิลป์ : ขั้นตอนวิธีการจัดทำดัชนีสำหรับข้อความไทยที่มีความผิดพลาด (INDEXING ALGORITHM FOR THAI TEXT WITH ERRORS) อ. ที่ปรึกษา : ผศ.ดร.สมชาย ประสิทธิ์จูตระกูล , 79 หน้า. ISBN 974-334-630-9.

วิทยานิพนธ์ฉบับนี้กล่าวถึงขั้นตอนวิธีการจัดทำดัชนีสำหรับข้อความไทยที่มีความผิดพลาด โดยมีจุดประสงค์ในการทำให้ดัชนีมีความสมบูรณ์มากขึ้นด้วยการเพิ่มคำที่ถูกต้องเข้าไปในดัชนีในกรณีที่ข้อความที่นำมาทำดัชนีมีความผิดพลาดปนอยู่ การจัดทำดัชนีที่นำเสนอนี้อาศัยคุณสมบัติ "ความเฉพาะตัว" ของสตริงซึ่งคือจำนวนครั้งของสตริงที่ปรากฏเป็นส่วนหนึ่งของคำในพจนานุกรม ขั้นตอนวิธีการจัดทำดัชนีแบ่งออกเป็นสามขั้นตอนคือ (1) หารายการของสตริงย่อยของข้อความที่ประกอบกันเป็นข้อความเดิมได้ โดยมีผลรวมของค่าของฟังก์ชัน (ที่มีค่าแปรตามค่าเฉพาะตัว) น้อยที่สุด (2) หาสตริงย่อยจากผลลัพธ์ที่ได้ในขั้นตอนแรกที่มีโอกาสสูงที่จะเกิดจากความผิดพลาดในข้อความ โดยพิจารณาจากค่าความเฉพาะตัวของสตริงย่อยที่เกินเกณฑ์ที่กำหนดไว้ และ (3) หาคำในพจนุกรมที่ใกล้เคียงกับคำหาได้จากการรวมสตริงย่อยของผลลัพธ์ในขั้นตอนที่สองกับสตริงข้างเคียงในข้อความ มาเป็นคำเพิ่มเติมในการจัดทำดัชนี จากผลการทดลองพบว่าสามารถเพิ่มความสมบูรณ์ให้กับดัชนีเดิมซึ่งไม่พิจารณาความผิดพลาดจาก 87% เป็น 94% ในขณะที่ลดความแม่นยำของดัชนีเดิมจาก 83% ลงเป็น 60%

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต .....
สาขาวิชา วิทยาศาสตร์คอมพิวเตอร์ ลายมือชื่ออาจารย์ที่ปรึกษา .....
ปีการศึกษา 2542 ลายมือชื่ออาจารย์ที่ปรึกษาร่วม .....

## 3971564021 : MAJOR COMPUTER SCIENCE

KEY WORD: INDEXING / ERRORS

VORAVAT VORASILP : INDEXING ALGORITHM FOR THAI TEXT WITH ERRORS.

THESIS ADVISOR : ASST. PROF. Dr. SOMCHAI PRASITJUTRAKUL, 79 pp. ISBN 974-334-630-9

This thesis presents an indexing algorithm for Thai text with errors. The algorithm utilizes string's "uniqueness" property which is defined to be the number of times that string appear as parts of words in a dictionary. There are three steps in the algorithm. First, we find a list of substrings which can be re-assembled to the original text and minimizes a function of substring uniquenesses. Second substrings of the list potentially caused by error are identified. This can be done by comparing a function of substring uniqueness to a preset threshold. Last, words in the dictionary which approximately match strings obtained by concatenating the potentially error-caused substrings and adjacent substrings are added in the index list. Experimental results showed that this algorithm can improve index completeness from 87% to 94% whiles decrease index precision from 83% to 60%.

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต .....

สาขาวิชา วิทยาศาสตร์คอมพิวเตอร์ ลายมือชื่ออาจารย์ที่ปรึกษา .....

ปีการศึกษา 2542 ลายมือชื่ออาจารย์ที่ปรึกษาร่วม .....

หอสมุดกลาง สถาบันวิทยบริการ จุฬาลงกรณ์มหาวิทยาลัย

# กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยความช่วยเหลืออย่างดียิ่งของ ผศ.ดร. สมชาย ประสิทธิ์จูตระกูล อาจารย์ที่ปรึกษาวิทยานิพนธ์ ซึ่งท่านได้ให้คำแนะนำและข้อคิดเห็นต่างๆ ในการวิจัยด้วยดีตลอดมาและตรวจแก้วิทยานิพนธ์ฉบับนี้อย่างละเอียด

ขอขอบพระคุณสถาบันพัฒนาวิทยาศาสตร์และเทคโนโลยีแห่งชาติ (สวทช.) ที่ให้การสนับสนุนทุนการศึกษาและทุนวิจัย

ขอขอบคุณห้องปฏิบัติการ Intelligence Systems Laboratory ที่เอื้อเฟื้ออุปกรณ์ในการทำงานวิจัย รวมทั้งพี่ ๆ และเพื่อน ๆ ที่ได้ให้คำปรึกษาและความช่วยเหลือในด้านต่าง ๆ ซึ่งทำให้การทำงานวิจัยเป็นไปอย่างราบรื่น

ท้ายนี้ ผู้วิจัยใคร่ขอกราบขอบพระคุณบิดา-มารดา ซึ่งสนับสนุนในทุกสิ่งทุกอย่างและให้กำลังใจแก่ผู้วิจัยเสมอมาจนสำเร็จการศึกษา

นายวรวัฒน์ วรศิลป์

มีนาคม 2543

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

# สารบัญ

หน้า

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

# สารบัญรูป

หน้า



สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

# สารบัญตาราง

หน้า



สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

# บทที่ 1

# บทนำ

## 1.1 ความเป็นมาและความสำคัญของปัญหา

ระบบสืบค้นข้อมูล (Information Retrieval System)[1] ได้ถูกพัฒนาขึ้นเพื่อช่วยให้สามารถทำการสืบค้นข้อมูลที่ต้องการได้อย่างรวดเร็วจากเอกสารที่มีเป็นจำนวนมาก ซึ่งระบบนี้ได้ถูกพัฒนาขึ้นเมื่อปี 1940 ในปัจจุบันหลาย ๆ มหาวิทยาลัย องค์กร ห้องสมุด รวมถึงอินเตอร์เนต (Internet) ได้นำระบบนี้เข้ามาใช้ เพื่อช่วยในการค้นหาข้อมูลต่าง ๆ

การจัดทำดัชนีคำเป็นวิธีการหนึ่งในการพัฒนาระบบสืบค้นข้อมูลให้ระบบสามารถค้นหาข้อมูลที่ต้องการได้อย่างรวดเร็ว

การจัดทำดัชนีคำนั้นก็คือการดึงคำสำคัญ(Keyword) ออกมาจากเอกสารแต่ละเอกสารในระบบและนำคำเหล่านั้นมาสร้างเป็นดัชนี เพื่อใช้อ้างอิงว่าคำสำคัญเหล่านั้นมีอยู่ในเอกสารใดบ้างในระบบ สำหรับการจัดเก็บดัชนีนั้นอาจจัดเก็บในรูปแฟ้มข้อมูลผกผัน (Inverted File)[1] ซึ่งจะกล่าวถึงโครงสร้างของแฟ้มข้อมูลแบบผกผันในภาคผนวก ก แต่ในเอกสารต่างๆ อาจเกิดความผิดพลาด เช่น พิมพ์ตก พิมพ์เกิน พิมพ์สลับ หรือ พิมพ์ผิด ดังตัวอย่างในตารางที่ 1.1 ซึ่งทำให้การสร้างดัชนีจากข้อความที่มีความผิดพลาดโดยวิธีการทำดัชนีแบบปกติซึ่งไม่คำนึงถึงความผิดพลาดจะส่งผลให้ได้ดัชนีที่ไม่สมบูรณ์ ตัวอย่างเช่น ในเอกสารหนึ่งผู้จัดทำต้องการพิมพ์คำว่า “เที่ยงธรรม” แต่พิมพ์ผิดเป็นคำว่า “เที่ยวธรรม” ซึ่งการจัดทำดัชนีปกติซึ่งไม่คำนึงว่าเอกสารนั้นมีความผิดพลาดหรือไม่ จะได้เพียงคำว่า “เที่ยว” และ “ธรรม” ไปจัดทำดัชนี ทำให้ดัชนีของเอกสารนี้ไม่มีคำว่า “เที่ยงธรรม” ตามที่ผู้จัดทำเอกสารต้องการ ดังนั้นเมื่อผู้ใช้ระบบทำการค้นหาเอกสารที่เกี่ยวข้องกับคำว่า “เที่ยงธรรม” ก็จะไม่ได้เอกสารดังกล่าวขึ้นมาเพราะเอกสารนั้นไม่มีคำว่า “เที่ยงธรรม” ในดัชนี เพราะมีแต่เพียงคำว่า “เที่ยว” และคำว่า “ธรรม”

**ตารางที่ 1.1 แสดงคำสำคัญที่ได้จากการจัดทำดัชนี**

| รูปแบบความผิดพลาด | ตัวอย่างข้อความ | ความผิดพลาด | คำสำคัญที่ได้จากขั้นตอนวิธีการจัดทำดัชนีทั่วไป | คำสำคัญที่ควรเพิ่มเข้าไป |
|---|---|---|---|---|
| พิมพ์ผิด | เที่ยวธรรม | พิมพ์ ง เป็น ว | เที่ยว , ธรรม | เที่ยงธรรม |
| พิมพ์เกิน | ประกกอบ | พิมพ์ ก เกิน | ประ ,กก,อบ | ประกอบ |
| พิมพ์ตก | บุเพสันนิวาส | พิมพ์ตกตัว พ | บุ , เพ , สันนิวาส | บุพเพสันนิวาส |
| พิมพ์สลับ | นายกรัฐนมตรี | พิมพ์สลับ ม กับ น | นายก, รัฐ , นม , ตรี | นายกรัฐมนตรี |

การจัดทำดัชนีสำหรับข้อความภาษาไทยก็มีความแตกต่างกับการจัดทำดัชนีสำหรับข้อความภาษาอังกฤษเนื่องจากการเขียนภาษาไทยไม่มีการเว้นวรรคระหว่างคำเหมือนภาษาอังกฤษ ทำให้มีปัญหาเรื่องขอบเขต เช่น "ตากลม" ซึ่งผู้เขียนต้องการสื่อ "ตาก" และ "ลม" แต่ผู้อ่านอาจเห็นเป็น "ตา" และ "กลม" ก็ได้ถ้าไม่พิจารณาข้อความรอบข้าง นอกจากนี้แล้วถ้าข้อความที่เข้ามามีความผิดพลาดอยู่ด้วยการที่ขอบเขตของคำไม่ชัดเจนจะทำให้การค้นหาความผิดพลาดและการแก้ไขความผิดพลาดทำได้ยากขึ้นด้วย การนำข้อความมาผ่านกระบวนการตัดคำก่อนและทำการหาข้อผิดพลาดและแก้ไข(Isolate Word Error Detection and Correction)โดยพิจารณาจากคำที่ได้จากโปรแกรมตัดคำก็ไม่ครอบคลุมเนื่องจากความผิดพลาดที่เกิดขึ้นเมื่อผ่านกระบวนการตัดคำแล้วอาจไม่แสดงให้เห็นความผิดพลาดเลย เช่น "สิงโตหลุดออกจากกรง" ถ้าโปรแกรมตัดคำตัดออกมาเป็น "สิ" "งง" "โต" "หลุด" "ออก" "จาก" "กรง" จะเห็นได้ว่าถ้าพิจารณาเป็นคำๆ ทุกคำจะอยู่ในพจนานุกรมหมดดังนั้นถ้ามองแบบแยกคำอาจกล่าวได้ว่าไม่มีความผิดพลาด แต่สำหรับการทำดัชนีถ้าเราคำนึงว่าให้ข้อความที่เข้ามามีความผิดพลาดได้ การพยายามหาข้อผิดพลาดและแก้ไขเพื่อสร้างคำที่ควรจะเพิ่มเข้าไปในดัชนี (ในกรณีนี้คำที่ควรเพิ่มเข้าไปคือ สิงโต)เพื่อให้ได้ดัชนีที่มีความสมบูรณ์มากขึ้นก็เป็นเรื่องที่น่าสนใจ และเป็นที่มาของวิทยานิพนธ์ฉบับนี้

## 1.2 วัตถุประสงค์ของวิทยานิพนธ์

เพื่อพัฒนาขั้นตอนวิธีการเพิ่มคำที่สมควรเพิ่มเข้าไปในดัชนีในกรณีที่คาดว่าข้อความที่เข้ามามีความผิดพลาด เพื่อทำให้ได้ดัชนีที่มีความสมบูรณ์มากขึ้น

## 1.3 ขอบเขตและเงื่อนไขของวิทยานิพนธ์

1. การออกแบบขั้นตอนวิธี เพิ่มคำที่สมควรเพิ่มเข้าไปในดัชนีในกรณีที่คาดว่าข้อความที่เข้ามามีความผิดพลาด เพื่อทำให้ได้ดัชนีที่มีความสมบูรณ์มากขึ้น โดยใช้ข้อมูลจากพจนานุกรมและข้อความรอบข้างโดยไม่ได้คำนึงถึงหลักไวยกรณ์

2. ความผิดพลาดที่เกิดขึ้นที่เกิดขึ้นเป็นแบบ ผิดที่เดียวใน 1 คำ ใน 4 รูปแบบคือ พิมพ์เกิน พิมพ์ตก พิมพ์ผิด และ พิมพ์สลับ และ ไม่ได้มีความผิดพลาดเกิดขึ้นถี่มากหรือเกิดติด ๆ กัน ซึ่งจะทำให้การวิเคราะห์หาความผิดพลาดและการแก้ไขความผิดพลาดมีความซับซ้อนมากเกินไป งานวิจัยนี้เป็นเพียงแนวทางเพื่อการพัฒนาขั้นตอนวิธีที่สามารถทนต่อความผิดพลาดที่มากขึ้นในภายหลัง

3. โปรแกรมทำงานภายใต้ระบบปฏิบัติการแบบ 32 บิต

4. การพัฒนาโปรแกรม จะใช้ภาษาระดับสูง (high level language) ที่มีความยืดหยุ่นสูง และพัฒนาโปรแกรมบนเครื่องไมโครคอมพิวเตอร์

5. รหัสภาษาไทยที่ใช้ จะใช้รหัสของสำนักงานมาตรฐานอุตสาหกรรม ( ส.ม.อ. )

6. พจนานุกรมที่ใช้เป็นพจนานุกรมราชบัณฑิต ที่นำมาจาก NECTEC

7. การพิจารณาประสิทธิภาพจะทำโดยนำข้อความมาสร้างความผิดพลาดและเปรียบเทียบดัชนีที่ได้จากข้อความต้นฉบับ กับ ข้อความที่ได้สร้างความผิดพลาดขึ้น โดยขั้นตอนวิธีที่พัฒนาขึ้นสำหรับทั้ง 2 ข้อความ

## 1.4 ขั้นตอนการวิจัย

1. ศึกษาขั้นตอนวิธีการแบ่งคำ
2. ศึกษาขั้นตอนวิธีที่ใช้ในการค้นหาความผิดพลาด
3. ศึกษาขั้นตอนวิธีที่ใช้ในการแก้ไขความผิดพลาด
4. ออกแบบและพัฒนาขั้นตอนวิธีการแบ่งคำที่เหมาะสมกับข้อความที่มีความผิดพลาด
5. ออกแบบและพัฒนาขั้นตอนวิธีเพื่อใช้ในการค้นหาความผิดพลาด
6. ออกแบบและพัฒนาขั้นตอนวิธีเพื่อใช้ในการแก้ไขข้อผิดพลาด
7. ออกแบบและพัฒนาขั้นตอนวิธีที่ใช้ในการทำดัชนี
8. ทดสอบขั้นตอนวิธีที่พัฒนาขึ้นว่าสามารถเพิ่มคำเข้าไปในดัชนีในกรณีที่เอกสารมีความผิดพลาดได้หรือไม่ และประสิทธิภาพเป็นอย่างไร
9. สรุปผลการวิจัยและข้อเสนอแนะ

## 1.5 ประโยชน์ที่คาดว่าจะได้รับ

1.ความรู้และแนวความคิดในการพัฒนาโครงสร้างข้อมูลและขั้นตอนวิธีที่ใช้ในการสืบค้นข้อมูล

2.ความเข้าใจถึงการทำงานของโครงสร้างข้อมูล และขั้นตอนวิธีในการสืบค้นคำ

3.ใช้เป็นแนวทางในการวิจัยเกี่ยวกับการออกแบบขั้นตอนวิธีการค้นหาความผิดพลาดและ การแก้ไขความผิดพลาด ในภาษาอื่น ๆ ที่มีลักษณะการเขียนประโยคคล้ายภาษาไทย

# บทที่ 2

# ทฤษฎีที่เกี่ยวข้อง

เนื่องด้วยภาษาไทยมีลักษณะการเขียนแตกต่างจากภาษาอังกฤษ โดยเฉพาะมีการเขียนคำติดกัน ทำให้การทำดัชนี การค้นหาความผิดพลาด และการแก้ไขความผิดพลาดไม่สามารถใช้วิธีการแบบเดียวกับที่ใช้กับภาษาอังกฤษได้การเขียนติดกันนั้นทำให้เกิดความกำกวม และยิ่งถ้ามีความผิดพลาดเกิดขึ้นด้วยแล้วส่วนที่ผิดพลาดอาจสามารถรวมได้กับคำใกล้เคียงซึ่งสามารถทำให้เกิดประโยคใหม่ที่ประกอบด้วยคำที่ถูกต้องทั้งหมดได้ซึ่งเป็นการยากในการที่จะบอกว่าประโยคมีความผิดพลาดอยู่ถ้าไม่พิจารณาอย่างรอบคอบ หรืออีกกรณีหนึ่งก็คือความผิดพลาดทำให้คำแตกออกเป็นหลายๆส่วนซึ่งแต่ละส่วนก็เป็นคำที่ถูกต้อง ถ้าใช้วิธีการธรรมดาก็จะไม่สามารถดึงคำลักษณะนี้ออกมาทำดัชนีได้ ในบทนี้จะกล่าวอย่างคร่าวๆ ถึงวิธีการที่ใช้จัดการกับความผิดพลาดซึ่งเป็นวิธีการทั่วไปที่ใช้กับภาษาอังกฤษ และจะชี้ให้เห็นว่าจะเกิดปัญหาอะไรขึ้นเมื่อมาใช้กับภาษาไทย

วิธีการที่ใช้ในการแก้ไขความผิดพลาดสำหรับภาษาอังกฤษ สามารถแยกออกเป็น 3 ส่วนตามกลุ่มของการทำวิจัย [2] คือ

1. การหาส่วนที่ไม่เป็นคำ ( Nonword error detection )
2. การแก้ไขคำผิดโดยไม่คำนึงถึงสภาพรอบข้าง ( Isolated-word error correction )
3. การแก้ไขคำผิดโดยขึ้นกับรูปประโยค ( Context-dependent word correction )

## 2.1 การหาส่วนที่ไม่เป็นคำ ( Nonword error detection )[2]

เป็นวิธีที่ใช้ในการตรวจหาคำผิด ซึ่งสำหรับภาษาอังกฤษสามารถทำได้ไม่ยากเนื่องจากมีขอบเขตของคำที่ชัดเจน

วิธีการที่นิยมใช้กับภาษาอังกฤษมีอยู่ 2 วิธีคือ

1) การเทียบกับพจนานุกรม ( dictionary lookup ) เป็นวิธีการที่ค่อนข้างตรงไปตรงมา คือใช้เทียบกับพจนานุกรม ถ้าคำไหนในข้อความไม่มีในพจนานุกรมก็ถือว่าเป็นคำผิด วิธีการนี้สามารถใช้กับภาษาอังกฤษได้เนื่องจากมีขอบเขตของคำที่ชัดเจน แต่สำหรับภาษาไทยซึ่งมีปัญหาเรื่องนี้ทำให้ใช้ได้ยาก การที่จะใช้โปรแกรมแบ่งคำประมวลผลก่อนเพื่อสร้างขอบเขตของคำขึ้นนั้นทำได้ แต่เนื่องจากประโยคภาษาไทยมีความกำกวมทำให้การแบ่งคำมีปัญหาและ

หอสมุดกลาง สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย



โปรแกรมแบ่งคำส่วนใหญ่ที่ใช้กับภาษาไทยจะเป็นโปรแกรมแบ่งพยางค์ที่พัฒนาขึ้นเพื่อใช้กับโปรแกรมจัดการเอกสารเพื่อใช้ในการตัดคำตรงท้ายบรรทัดซึ่งไม่เหมาะสมเท่าที่ควร อีกประการหนึ่งก็คือโปรแกรมแบ่งพยางค์เหล่านั้นเป็นโปรแกรมที่แบ่งคำโดยใช้กฎและพัฒนาโดยมีพื้นฐานว่าข้อความที่เข้ามาไม่มีความผิดพลาด ผู้วิจัยได้ทดลองใช้โปรแกรมประเภทนี้โปรแกรมหนึ่ง (thaisep) กับข้อความที่มีความผิดพลาด ผลลัพธ์ที่ได้ออกมาไม่เหมาะสมสำหรับทำการประมวลผลในขั้นต่อไปตามแนวคิดของผู้วิจัย เนื่องจากเมื่อมีความผิดพลาดแล้วผลการแบ่งคำส่วนที่ผิดก็จะผิดแถมยังสร้างผลกระทบไปยังคำข้างเคียงที่เป็นคำที่ถูกต้องอยู่แล้วค่อนข้างมาก ตัวอย่างเช่น "สิงโตตทำลายกรงออกมา" ซึ่งเมื่อผ่านโปรแกรมดังกล่าวจะได้ผลลัพธ์ของการแบ่งคำคือ "สิง" , "โตตทำ" , "ลายกรงออก" , "มา" อย่างไรก็ดีผลกระทบข้างเคียงนี้จะเกิดขึ้นระยะหนึ่งแล้วหมดไป แต่ก็ทำให้มีปัญหาที่ต้องพิจารณาเพิ่มขึ้น ทำให้ผู้วิจัยได้พัฒนาโปรแกรมแบ่งคำขึ้นใหม่ซึ่งจะกล่าวถึงในบทที่ 4

2) การวิเคราะห์ n-gram ( n-gram base analysis ) n-gram เป็นโครงสร้างข้อมูลที่ใช้ในการเก็บข้อมูลการมีเกิดขึ้นของสตริงต่างๆ ที่ปรากฏเป็นส่วนหนึ่งของในคำพจนานุกรม การเก็บการมีเกิดขึ้นของสตริงอาจเก็บแบบ มี หรือ ไม่มี ( binary ) กล่าวคือถ้าสตริงปรากฏเป็นส่วนหนึ่งของในคำพจนานุกรมก็จะเก็บว่ามีเกิดขึ้น(1) ในขณะที่ถ้าสตริงไม่ปรากฏเป็นส่วนหนึ่งของในคำพจนานุกรมก็จะเก็บว่าไม่มีเกิดขึ้น(0) หรืออาจเก็บเป็นความถี่หรือจำนวนครั้งที่สตริงนั้นๆ ปรากฏเป็นส่วนหนึ่งของในคำพจนานุกรม ยกตัวอย่างเช่น 2-gram ( bi-gram ) ก็จะมีรูปร่างเป็นตารางที่มีจำนวนแถว และ จำนวนคอลัมน์เท่ากับจำนวนอักขระในภาษานั้นๆ เช่น ภาษาอังกฤษก็จะเป็นตารางขนาด 26×26 สำหรับภาษาไทยตารางก็จะมีขนาดใหญ่กว่า เนื่องจากมีตัวอักษรมากกว่า มีสระ วรรณยุกต์ และอักขระพิเศษซึ่งมีใช้กับข้อความประเภทโคลงกลอน ถ้าใช้ทั้งหมดรวมเป็น 70 ตัวเมื่อนำมาสร้าง 2-gram ก็จะเป็นตารางขนาด 70×70 ดังรูปที่ 2.1 (เก็บแบบ มี หรือ ไม่มี โดยสมมติว่าในพจนานุกรมมีเพียงคำว่า กระรอก กระบอก หน้าไม้)

การใช้ n-gram ในการค้นหาความผิดพลาดสามารถทำได้โดยการตัดคำที่ต้องการทดสอบออกเป็นส่วน ๆ ถ้ามีส่วนใดส่วนหนึ่งเป็นสตริงที่ไม่ถูกต้องเมื่อเทียบกับ n-gram แสดงว่าคำนั้นมีความผิดพลาด ความสามารถในการหาความผิดพลาดของการใช้วิธีนี้ว่าจะค้นพบความผิดพลาดได้มากน้อยเพียงไรขึ้นอยู่กับค่า n ถ้ามากเปอร์เซ็นต์การพบความผิดพลาดก็จะสูงขึ้นแต่ก็จะใช้พื้นที่ในการเก็บ n-gram เพิ่มขึ้นตามไปด้วย การจะนำวิธีนี้มาใช้กับข้อความภาษาไทยจะทำได้ก็ต่อเมื่อการแบ่งคำถูกต้องเหมือนกับวิธีเทียบพจนานุกรมแต่เราสามารถใช้ n-gram ช่วยทดสอบอะไรบางอย่างได้ เช่นถ้านำข้อความที่ยังไม่ได้ทำการแบ่งคำมาตัดเป็นส่วน ๆ แล้วเทียบกับตารางถ้าส่วนไหนไม่เป็นสตริงที่ถูกต้อง แสดงว่าส่วนนั้นอาจเป็นรอยต่อของคำ 2 คำ หรืออาจเกิดจากความผิดพลาดจริง

| | ก | ... | บ | ... | น | ... | ม | ... | ร | ... | ห | ... | อ | ... | ะ | ... | า | ... | ไ | ... | ้ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ก | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| บ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| น | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ม | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ร | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ห | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| อ | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ะ | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| า | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ไ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ้ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

**รูปที่** 2.1 **ตัวอย่าง** n-gram table

2.2 **การแก้ไขคำผิดโดยไม่คำนึงถึงสภาพรอบข้าง** ( Isolated-word error correction )[2]

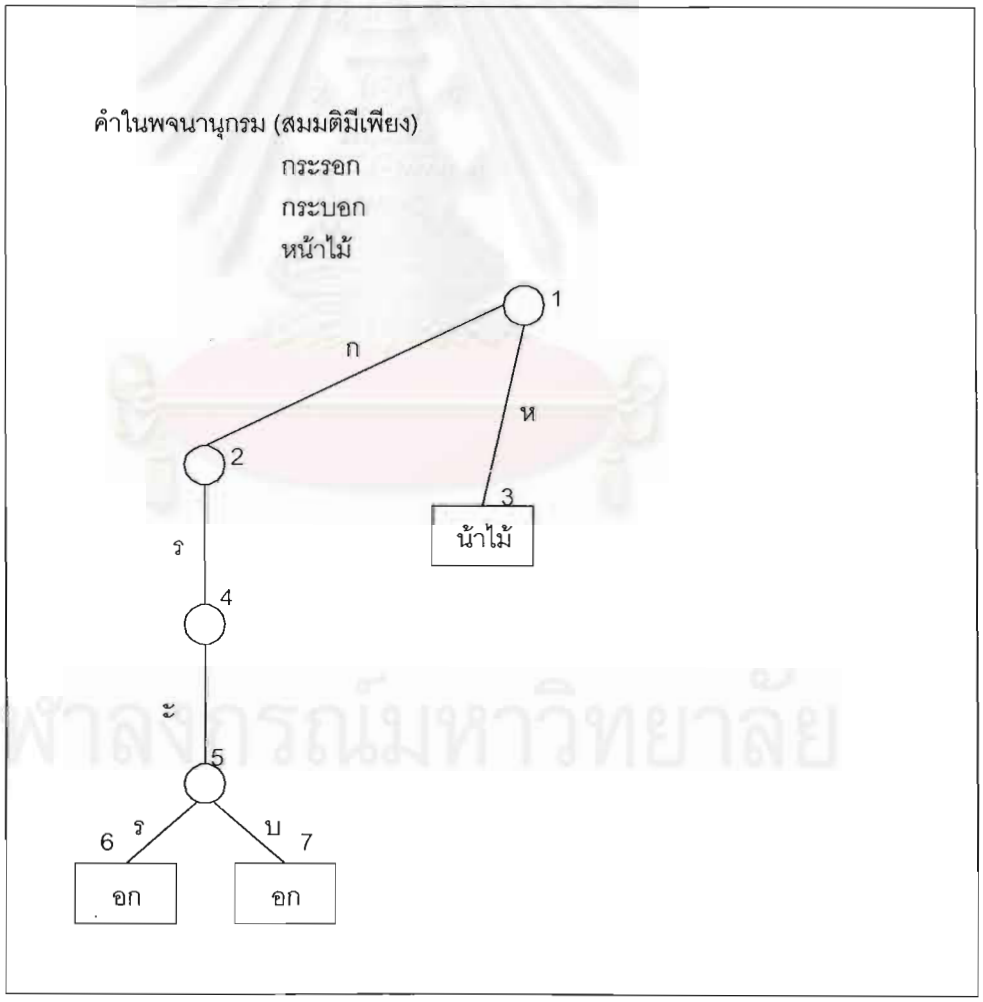
เป็นวิธีที่ใช้แก้ไขคำผิดที่ได้จากกระบวนการหาส่วนที่ไม่เป็นคำ สำหรับภาษาอังกฤษซึ่งมีขอบเขตของคำที่ผิดชัดเจนสามารถทำได้ไม่ยากเช่น การใช้ minimum edit distance ซึ่งคือการหาคำในพจนานุกรมที่ใกล้เคียงกับคำที่ผิดมากที่สุด กล่าวคือคำในพจนานุกรมที่ได้จากการแก้ไขคำที่ผิดน้อยที่สุด เช่น "กระาก" จะได้คำว่า "กระชาก" ได้จากการแก้ไขโดยการเติม "ช" , "กราก" ได้จากการลบ "ะ" วิธีนี้ใช้ได้ดีถ้ารู้ขอบเขตของคำที่ผิดอย่างชัดเจน แต่ดังที่ได้กล่าวไปแล้วการจะรู้ขอบเขตของคำในประโยคภาษาไทยโดยเฉพาะประโยคที่มีความผิดพลาดเป็นเรื่องยาก การนำข้อความไปผ่านการแบ่งคำเพื่อหาขอบเขตของคำก่อนบางครั้งก็ทำให้หาที่ผิดคลาดเคลื่อน(เห็นเพียงบางส่วนของคำผิดที่ขีดเส้นใต้ไว้) เช่น "กรมการปกครอ" เมื่อให้โปรแกรมแก้ไขคำผิดที่ไม่คำนึงถึงสภาพรอบข้างแนะนำว่าควรแก้ไขอย่างไรก็อาจได้คำ "ครือ" ,"ครอก", "ครอง" ,"ครอบ" , "กรอ" และ "กรอง" ซึ่งไม่ค่อยเหมาะสมส่วนที่ผิดนั้นอาจเป็นเพียงส่วนหนึ่งของคำที่ผิด

2.3 **การแก้ไขคำผิดโดยขึ้นกับรูปประโยค** ( Context-dependent word correction )[2]

เป็นการแก้ไขคำผิดโดยการวิเคราะห์ถึงระดับไวยกรณ์ของประโยคซึ่งค่อนข้างซับซ้อน ซึ่งวิทยานิพนธ์นี้ยังไม่ได้ใช้หลักการนี้ จึงไม่ขอกล่าวถึงหลักการนี้ อย่างไรก็ดีวิธีการนี้ก็เป็นวิธีการหนึ่งที่จะสามารถนำมาปรับปรุงกระบวนการการทำดัชนีสำหรับข้อความไทยที่มีความผิดพลาดได้

2.4 **โครงสร้างข้อมูลแบบทรัย**(trie)[1]

ทรัย เป็น digital tree ชนิดหนึ่งซึ่งเมื่อนำมาใช้ในการเก็บพจนานุกรมแล้วจะสามารถใช้ในการค้นหาคำที่ต้องการว่ามีอยู่ในพจนานุกรมหรือไม่โดยใช้เวลาเท่ากับ O(m) โดยที่ m เป็นความยาวของคำที่ต้องการหา และ ทรัย ยังสนับสนุนการหาการค้นหาอักขระเติมหน้า(prefix)ด้วย ตัวอย่างของทรัยแสดงในรูปที่ 2.2




**รูปที่** 2.2 **แสดงการเก็บคำในพจนานุกรมบนโครงสร้างข้อมูลแบบทรัย**

2.5 **การจับคู่แบบประมาณบนโครงสร้างข้อมูลแบบทรัย** (Approximate String Matching on trie)[6]

เป็นการหาสตริงที่ต้องการจากเซตของคำที่เก็บบนโครงสร้างข้อมูลแบบทรัยโดยประมาณ กล่าวคืออนุญาตให้มีความผิดพลาดเกิดขึ้นได้ k ตำแหน่งในรูปแบบที่กำหนด เช่น คิดว่า สตริงที่ต้องการหาเกิดจากการพิมพ์ตก พิมพ์ผิด พิมพ์เกิน หรือ พิมพ์สลับ

ตัวอย่างรหัสเทียม(pseudo code)สำหรับการจับคู่แบบประมาณแสดงดังรูปที่ 2.3 โดยคำนึงเฉพาะความผิดพลาดแบบพิมพ์ตกซึ่งมีลักษณะการทำงานแบบเรียกซ้ำ (recursive) ซึ่งพารามิเตอร์ที่ใช้ในโปรแกรมฟังก์ชั่นมีดังนี้

| | |
|---|---|
| current_node | เป็นตัวชี้ตำแหน่งของทรัยที่จะใช้ในการเปรียบเทียบตัวอักษรที่กำลังพิจารณา |
| err_avail | เป็นตัวแปรที่ใช้เก็บจำนวนความผิดพลาดที่ยังยอมให้เกิดขึ้นได้อีก |
| pos_o | เป็นตำแหน่งของตัวอักษรที่กำลังพิจารณาในลำดับตัวอักษรที่ต้องการทำการจับคู่แบบประมาณ |
| pos_a | เป็นตำแหน่งของตัวอักษรในลำดับตัวอักษรผลลัพธ์ระหว่างการทำการจับคู่แบบประมาณ |
| str[] | เป็นลำดับตัวอักษรที่ต้องการทำการจับคู่แบบประมาณ |
| w[] | เป็นลำดับตัวอักษรผลลัพธ์ระหว่างการทำการจับคู่แบบประมาณ |

```
approx_deletion_error(TRIE_NODE *current_node,int err_avail,int pos_o,int pos_a,
char str[],char w[])
{
    if (pos_o == strlen(str))
        add w to approximate match word list
   else
        if availble branch str[pos_o] on current_node
        {   // path 1
            w[pos_a] = str[pos_o];
            approx_deletetion_error(current_node->branch[str[pos_o]],err_avial,
                                    pos_o+1,pos_a+1 ,str,w);
        }
        if (err_avail > 0)
            for any other branch c  of current node ( c != str[pos])
            {    // path 2
                w[pos] =  c;
                approx_deletetion_error(current_node->branch[c],err_avial-1,
                                        pos_o,pos_a+1,str,w);
            }

}
```

**รูปที่ 2.3 แสดงตัวอย่างรหัสเทียมสำหรับการจับคู่แบบประมาณ**

ตัวอย่าง แสดงการจับคู่แบบประมาณของ “กระอก” บนทรัยในรูปที่ 2.2 โดยใช้รหัสเทียมในรูปที่ 2.3 โดยแสดงการทำงานในรูปแบบของต้นไม้ดังแสดงในรูปที่ 2.4 ซึ่งจะได้ “กระรอก” และ “กระบอก” เป็นผลลัพธ์

โดยแต่ละโหนดในต้นไม้จะแสดงค่าพารามิเตอร์ในแต่ละขั้นของการประมวลผลและกิ่งของต้นไม้แสดงเส้นทางในการประมวลผล เช่น โหนดเริ่มต้นในต้นไม้แสดงการประมวลผลจะแสดงค่าพารามิเตอร์ก่อนการประมวลผลซึ่งสามารถอธิบายได้ดังนี้ ค่า currrent_node = 1 เป็นการตั้งค่าตำแหน่งของโหนดในทรัยในรูปที่ 2.2 ให้เป็นโหนดบนสุด err_avail = 1 ตั้งค่าตัวนับความผิดพลาดที่ยอมให้เกิดได้ pos_o = 0 ตั้งค่าตำแหน่งตัวอักษรที่จะใช้พิจารณาในสตริงที่ต้องการทำการจับคู่แบบประมาณให้เป็นตำแหน่งเริ่มต้น(ตำแหน่งของตัวอักษรเริ่มนับจาก 0 เป็นตำแหน่งแรก) pos_a = 0 ตั้งค่าตำแหน่งตัวอักษรในสตริงที่ใช้ในการเก็บผลลัพธ์ของการทำการจับคู่แบบประมาณให้เป็นตำแหน่งเริ่มต้น w = “” ตั้งค่าสตริงที่ใช้เก็บผลลัพธ์ของการจับคู่แบบประมาณให้เป็นสตริงว่าง เมื่อทำการประมวลผลตามรหัสเทียมจะเห็นว่าค่าพารามิเตอร์จะตรงกับเงื่อนไขของทั้ง 2 เส้นทาง ดังแสดงในรูปที่ 2.4 จะเห็นว่าจากโหนดเริ่มต้นจะมีกิ่ง 2 กิ่งซึ่งแสดงการประมวลผลในทั้ง 2 เส้นทางตามรหัสเทียมโดยโหนดใหม่ก็จะมีค่าพารามิเตอร์ที่ใช้ในการประมวลผลต่างไป เช่น เมื่อพิจารณาเส้นทางที่ 1 ในรหัสเทียมจะเห็นว่ามีการเขียนตัวอักษรลงใน w และทำการเรียกตัวเองโดยเปลี่ยนค่าพารามิเตอร์ต่างๆ ซึ่งจะเห็นว่าดโหนดที่เกิดขึ้นจากการประมวลผลมีพารามิเตอร์ต่างออกไปดังแสดงในรูปที่ 2.4 ข้อสังเกตุอีกจุดหนึ่งคือการประมวลผลเส้นทางที่ 2 อาจมีได้หลายเส้นทางขึ้นอยู่กับว่ามีคำในพจนานุกรมที่ใกล้เคียงกับสตริงที่ต้องการทำการจับคู่แบบประมาณมากน้อยเพียงไร

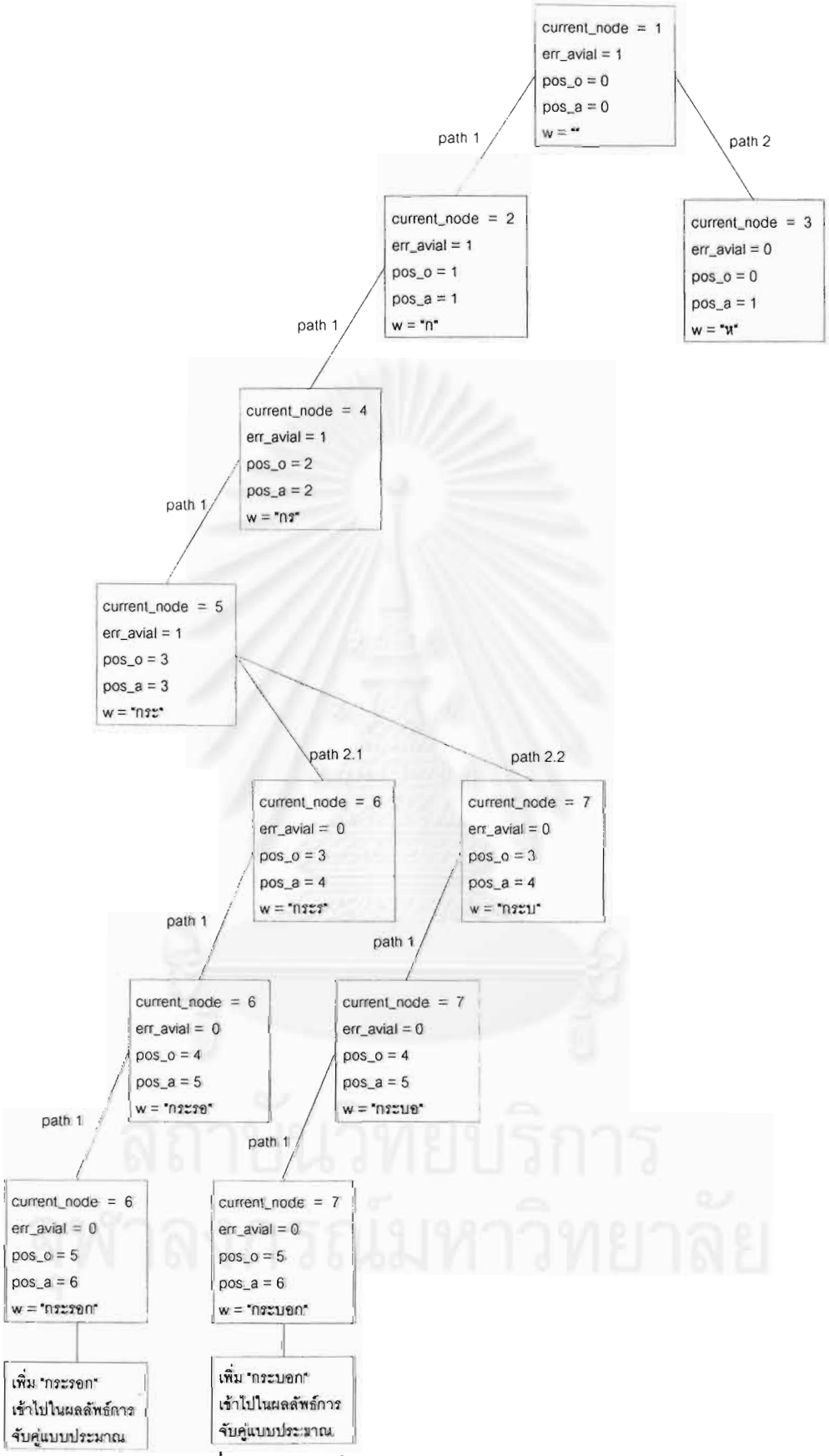
## 2.6 การหาเส้นทางสั้นสุด(Shortest path)[5]

การหาเส้นทางสั้นสุดเป็นวิธีการที่ใช้ในการหาเส้นทางที่สั้นสุดใน กราฟ G (V,E) ที่กำหนดวิธีการนี้สามารถนำมาประยุกต์ใช้ในการหาคำตอบที่เหมาะสมสำหรับปัญหาที่สามารถแปลงเป็นกราฟได้

## 2.7 การดึงคำด้วยพจนานุกรม

การดึงคำด้วยพจนานุกรมที่จะกล่าวถึงต่อไปนี้เป็นวิธีการที่พบในวิทยานิพนธ์ของวิฑูรย์ กัลยาณวัฒน์ เรื่องระบบการค้นคืนข้อความภาษาไทยโดยใช้แฟ้มข้อมูลผกผัน [4] ซึ่งเป็นวิธีการหนึ่งที่เหมาะสมสำหรับการดึงคำเพื่อนำมาทำดัชนี ซึ่งวิธีการดังกล่าวสามารถจัดการกับคำที่ไม่ปรากฏในพจนานุกรมได้ด้วย รายละเอียดคร่าวๆ ของวิธีการมีเป็นขั้นๆดังนี้

1. หาคำที่ยาวที่สุดต่างๆ ที่มีในพจนานุกรมที่ปรากฏในข้อความ




**รูปที่ 2.4 แสดงตัวอย่างการจับคู่แบบประมาณบนทรัย**

2. สร้างกราฟที่แทนการติดต่อกันและการทับกันของคำต่างๆ ในข้อความ โดยมีน้ำหนักของเส้นเชื่อมแทนลักษณะการต่อกันและการทับกันของคำต่างๆเหล่านั้น(สำหรับรายละเอียดของการให้น้ำหนักเส้นเชื่อมสามารถหาอ่านเพิ่มได้จาก [4])

3. หาเส้นทางที่สั้นที่สุดในกราฟนี้ ซึ่งแทนกลุ่มที่เล็กสุดของคำในข้อความ ที่เมื่อเลือกแล้วจะลดจำนวนสายอักขระที่ไม่รู้จักให้ปรากฏขึ้นเป็นจำนวนน้อยที่สุด

4. สายอักขระย่อยเหล่านี้จะถูกเทียบกับพยางค์ต่างๆ ในข้อความโดยการใช้ขั้นตอนวิธีการแบ่งพยางค์แบบใช้กฎ

ผลที่ได้คือคำต่างๆที่ได้จากเส้นทางสั้นสุดของกราฟ และพยางค์ต่างๆ ที่ได้จากการเทียบกับสายอักขระย่อยที่ไม่เป็นคำที่รู้จัก จะเป็นกลุ่มของคำสำคัญในการจัดทำดัชนีของข้อความที่ได้รับ

ตัวอย่าง ข้อความ "นายเจมส์มาร์ตินต้องการผลิตรายการโทรทัศน์" ดังนี้

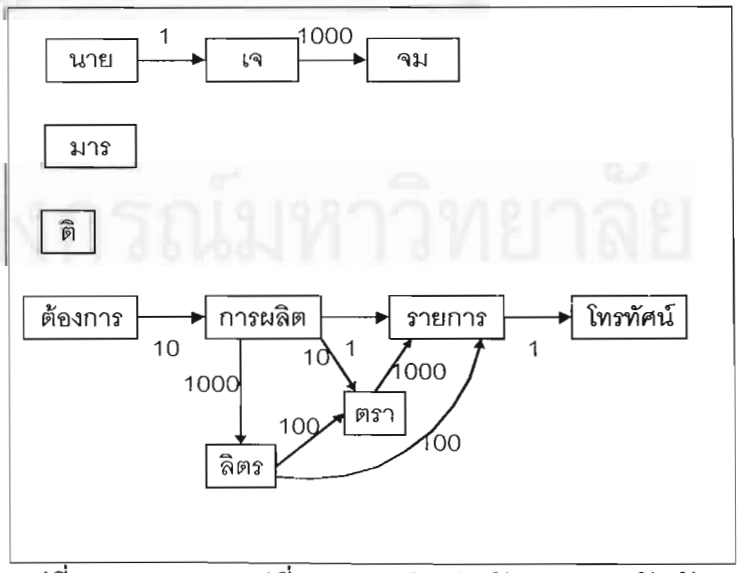
จากขั้นตอนที่ 1 จะได้ คำ นาย , เจ , จม , มาร , ติ , ต้องการ , การผลิต , ลิตร , ตรา , รายการ , โทรทัศน์

จากขั้นตอนที่ 2 จะได้กราฟที่แทนการติดต่อกันและการทับกัน ดังรูปที่ 2.5

จากขั้นตอนที่ 3 หาเส้นทางสั้นสุดจากกราฟที่ได้ในขั้นตอนที่ 2

จากขั้นตอนที่ 4 จะได้ นาย , ต้องการ , การผลิต , รายการ , โทรทัศน์ , ต้อง , ผลิต เป็นคำและ นายเจมส์มาร์ติน เป็นคำที่ไม่รู้จัก

ซึ่งวิธีการในการดึงคำนี้ได้นำมาประยุกต์ใช้ในการจัดทำดัชนีสำหรับข้อความไทยที่มีความผิดพลาดดังจะกล่าวในบทที่ 4




**รูปที่ 2.5 แสดงกราฟที่แทนการติดต่อกันและการทับกัน**

# บทที่ 3

## การออกแบบขั้นตอนวิธีการทำดัชนีสำหรับข้อความภาษาไทยที่มีความผิดพลาด

จุดประสงค์หลักของการทำดัชนีสำหรับข้อความภาษาไทยที่มีความผิดพลาดนั้นคือการดึงคำสำคัญออกจากข้อความเพื่อนำไปทำดัชนีโดยเพิ่มความสมบูรณ์ของดัชนี ในการที่จะเพิ่มความสมบูรณ์ของดัชนีได้นั้นจำเป็นที่จะต้องทำการค้นหาความผิดพลาดและทำการวิเคราะห์ความผิดพลาดที่ค้นพบเพื่อเพิ่มคำสำคัญที่ได้แก้ไขแล้วอย่างเหมาะสมเข้าไปในดัชนี อย่างไรก็ดีภาษาไทยมีปัญหาเรื่องขอบเขตของคำซึ่งทำให้ไม่สามารถใช้วิธีค้นหาความผิดพลาดในลักษณะเดียวกับภาษาอังกฤษซึ่งเขียนแยกคำได้ทำให้จำเป็นต้องคิดค้นวิธีการค้นหาความผิดพลาดขึ้นมาใหม่ ซึ่งวิธีการที่คิดขึ้นนี้ทำโดยใช้ค่าความเฉพาะตัวของสตริงที่ผู้วิจัยได้คิดค้นขึ้นมาเพื่อใช้เป็นเครื่องมือช่วยในการวิเคราะห์

ค่า "ความเฉพาะตัว" $U(s)$ ของสตริง $s$ คือ จำนวนครั้งของ $s$ ที่ปรากฏเป็นส่วนหนึ่งของคำในพจนานุกรม

ตัวอย่าง $U$ ("นิวาส" ) = 4

เพราะ "นิวาส" ปรากฏใน "นิวาส" , "บุพเพนิวาสานุสติญาณ" , "บุพเพสันนิวาส" และ "สันนิวาส"

เราสามารถใช้ค่าความเฉพาะตัวของสตริงนี้กับสตริงต่างๆ ที่ได้จากการแบ่งข้อความออกเป็นสตริงหรือคำ สตริงใดที่มีค่าความเฉพาะตัวสูงนั้นหมายถึงสตริงนั้นเป็นส่วนประกอบของคำหลายๆคำในพจนานุกรม นั่นหมายถึงว่าสตริงนั้นอาจยังเป็นคำที่ไม่สมบูรณ์ซึ่งอาจเกิดจากมีความผิดพลาดขึ้นในคำทำให้โปรแกรมแบ่งคำแบ่งคำที่ผิดพลาดซึ่งควรเป็นคำหนึ่งคำออกเป็นคำย่อยหลายๆคำ ตัวอย่างเช่น "บุเพสันนิวาส" เมื่อผ่านการแบ่งคำจะได้ออกมาเป็น "บุ" , "เพ" และ "สันนิวาส" ดังนั้นการใช้ค่าความเฉพาะตัวในการหาความผิดพลาดนั้นสามารถทำได้ง่ายๆ โดยนำค่าความเฉพาะตัวของสตริงไปผ่านฟังก์ชั่นซึ่งฟังก์ชั่นนี้จะทำการปรับให้ค่าความเฉพาะตัวมีความเหมาะสมยิ่งขึ้นเนื่องจากค่าความเฉพาะตัวนี้จะลดลงเมื่อจำนวนตัวอักษรในสตริงเพิ่มขึ้นเป็นปรกติอยู่แล้ว ดังนั้นใช้ฟังก์ชั่นนี้จึงมีหน้าที่ปรับให้สตริงที่มีค่าความเฉพาะตัวเท่ากันแต่มีความยาวเป็นตัวอักษรต่างกันมีความหมายต่างกัน ค่าที่ได้จากฟังก์ชั่นจะใช้เป็นตัวชี้ว่าสตริงที่พิจารณานั้นน่าจะเป็นความผิดพลาดหรือไม่โดยการเปรียบเทียบกับเกณฑ์ที่กำหนดซึ่งผู้ใช้สามารถปรับเกณฑ์นี้ได้ตามความเหมาะสม ถ้าค่าที่ได้นั้นมากกว่าเกณฑ์ที่กำหนดไว้ก็ให้ถือว่าสตริงนั้นน่าจะเป็นความผิดพลาด นอกจากนี้ค่าความเฉพาะตัวของสตริงไปประยุกต์ใช้กับการแบ่งคำด้วยดังจะกล่าวต่อไป

สำหรับขั้นตอนหลักของการทำดัชนีสำหรับข้อความภาษาไทยที่มีความผิดพลาดประกอบด้วย การดึงคำแบบปกติซึ่งทำโดยไม่คำนึงถึงว่าข้อความมีความผิดพลาดปนอยู่ และการดึงคำเพิ่มเติมสำหรับกรณีที่คาดว่าในข้อความมีความผิดพลาดปนอยู่ พิจารณา รูปที่ 3.1 ซึ่งแสดงขั้นตอนหลักของการทำดัชนีสำหรับข้อความภาษาไทยที่มีความผิดพลาดโดยใช้ตัวอย่าง ข้อความ "นักบนนอนตากลม" ซึ่งเป็นข้อความที่มีความผิดพลาดโดยข้อความที่ถูกต้องคือ "นักบินนอนตากลม"

1. การดึงคำแบบปกติซึ่งทำโดยไม่คำนึงถึงว่าข้อความมีความผิดพลาดปนอยู่ แบ่งเป็นขั้นตอนย่อยๆ คือ

1.1 การแบ่งคำ เนื่องจากภาษาไทยเขียนโดยไม่มีการเว้นวรรคระหว่างคำทำให้เกิดปัญหาเรื่องขอบเขตของคำดังนั้นจึงจำเป็นต้องใช้ขั้นตอนนี้ในการแบ่งข้อความออกเป็นคำเพื่อความสะดวกต่อการประมวลผลในขั้นต่อๆไป การแบ่งคำนี้ทำโดยดึงคำออกจากข้อความโดยการเลือกคำยาวสุดในแต่ละตำแหน่งของข้อความ ถ้าคำที่ดึงออกมามีส่วนที่ทับกันก็สร้างคำเพิ่มเติมขึ้นมาเพราะเป็นไปได้ว่าส่วนที่ทับกันนั้นอาจแบ่งคำได้หลายแบบเนื่องจากความกำกวม การสร้างคำเพิ่มเติมนี้ก็เพื่อให้สามารถเลือกการแบ่งคำหนึ่งรูปแบบมาประมวลผลได้ในกรณีที่มีความกำกวม หลังจากนั้นให้น้ำหนักกับ สตริงแต่ละสตริงโดยน้ำหนักของสตริงนั้นจะเป็นฟังก์ชั่นของค่าความเฉพาะตัวของสตริง จากนั้นนำไปสร้างกราฟและหาเส้นทางสั้นสุดและใช้เส้นทางสั้นสุดนั้นแบ่งคำ การใช้เส้นทางสั้นสุดนั้นจะทำให้ได้คำที่มีน้ำหนักน้อย หรือ มีค่าความเฉพาะตัวน้อยเป็นผลลัพธ์ในการแบ่งคำ ซึ่งนั่นก็คือการแบ่งคำจะเลือกคำที่มีโอกาสผิดพลาดน้อยนั่นเอง

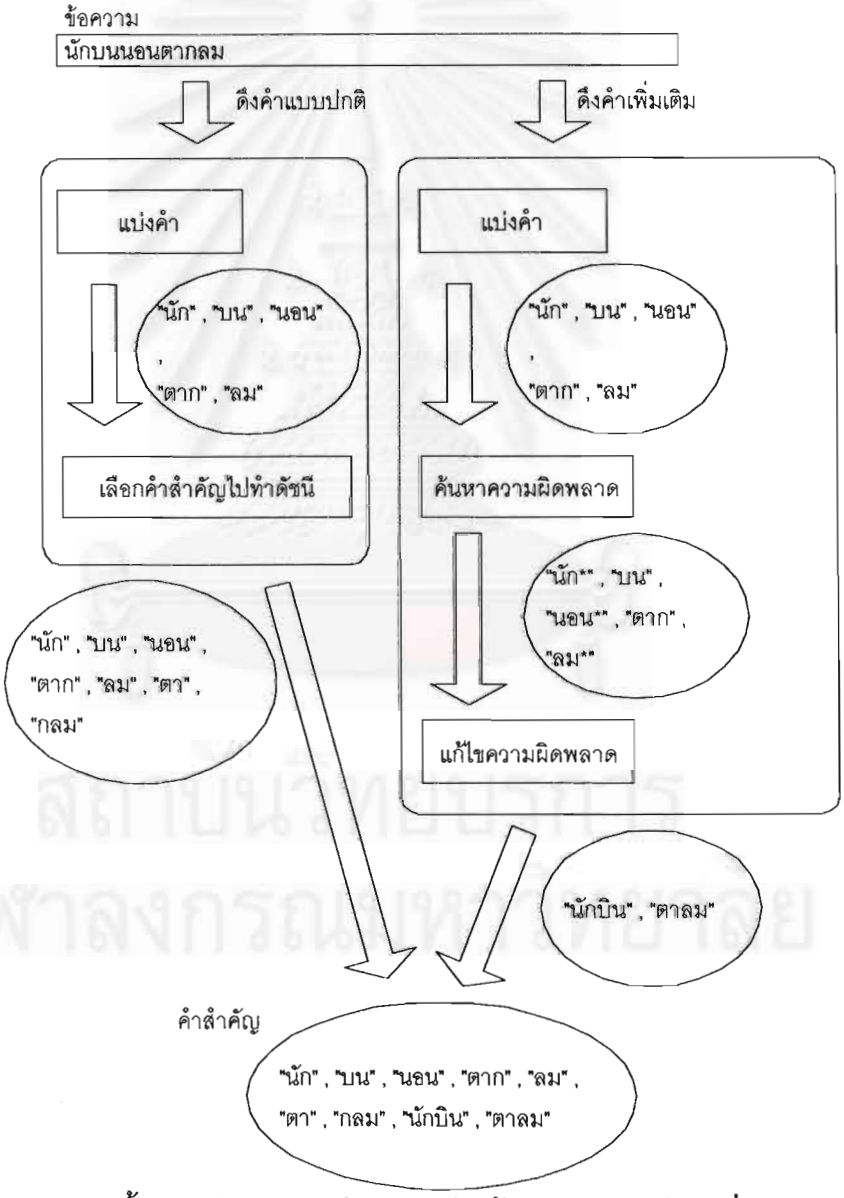
1.2 เลือกคำสำคัญไปทำดัชนี โดยขั้นตอนนี้จะนำคำที่เป็นผลลัพธ์จากการแบ่งคำที่ผ่านเงื่อนไขซึ่งเป็นเกณฑ์ที่ใช้ในการคัดคำที่เหมาะสมไปทำดัชนี และในกรณีที่พบว่าประโยคมีความกำกวม(ประโยคสามารถทำการแบ่งคำได้หลายแบบ เช่น "ตากลม" สามารถแบ่งได้ 2 แบบ คือ "ตา" กับ "กลม" หรือ "ตาก" กับ "ลม") ก็ให้เพิ่มคำที่จะได้จากการแบ่งคำในรูปแบบอื่นเข้าไปในดัชนีด้วย

2. การดึงคำเพิ่มเติมสำหรับกรณีที่คาดว่าในข้อความมีความผิดพลาดปนอยู่ แบ่งเป็นขั้นตอนย่อยๆ คือ

2.1 การแบ่งคำ จะเหมือนกับ 1.1

2.2 การค้นหาความผิดพลาด ขั้นตอนนี้จะใช้ในการค้นหาความผิดพลาดที่ปนอยู่ในข้อความโดยใช้ข้อมูลที่ได้รับจากการแบ่งคำ โดยเปรียบเทียบค่าน้ำหนักของสตริงแต่ละสตริงที่ได้จากการแบ่งคำว่าเกินค่าที่ผู้ใช้งานตั้งไว้หรือไม่ถ้าเกินก็ถือว่าเป็นความผิดพลาดและเก็บข้อมูลไว้ใช้ในการประมวลผลในขั้นตอนต่อไปซึ่งในตัวอย่างคำที่ถูกพิจารณาว่าเป็นความผิดพลาดคือ "นัก" , "นอน" , "ลม"

2.3 การแก้ไขความผิดพลาด ขั้นตอนนี้จะทำการเพิ่มคำที่ได้จากการแก้ไขความผิดพลาดเข้าไปในดัชนีในกรณีที่พบว่าในข้อความมีความผิดพลาดปนอยู่ โดยนำคำที่คาดว่ามีความผิดพลาดที่ได้จากขั้นการค้นหาความผิดพลาด มาทำการวิเคราะห์ว่าเป็นไปได้ไหมที่จะเป็นความผิดพลาดโดยการรวมคำที่สงสัยกับคำข้างเคียงและทำการจับคู่แบบประมาณถ้าได้ผลลัพธ์ออกมาเป็นคำที่อยู่ในพจนานุกรมก็นำคำที่ได้มานั้นเพิ่มเข้าไปในดัชนี เช่นคำว่า “นักบิน” ในตัวอย่างแต่พิมพ์ผิดเป็น ”นักบน” ซึ่งเมื่อผ่านการแบ่งคำจะได้ “นัก” และ “บน” ซึ่ง “นัก” ถูกพิจารณาว่าเป็นความผิดพลาด ขั้นการแก้ไขความผิดพลาดนี้ก็จะทำการรวมคำว่า “นัก” ซึ่งคิดว่าเป็นความผิดพลาดเข้ากับ “บน” ซึ่งเป็นคำข้างเคียงแล้วนำไปทำการจับคู่แบบประมาณซึ่งจะได้ผลลัพธ์เป็นคำ “นักบิน” และทำการเพิ่มคำนี้เข้าไปในดัชนี




**รูปที่ 3.1 แสดงขั้นตอนวิธีการทำดัชนีสำหรับข้อความภาษาไทยที่มีความผิดพลาด**

# บทที่ 4

## การพัฒนาขั้นตอนวิธีการทำดัชนีสำหรับข้อความภาษาไทยที่มีความผิดพลาด

เนื่องจากภาษาไทยเขียนโดยไม่มีการเว้นวรรคระหว่างคำทำให้เกิดปัญหาเรื่องขอบเขตของคำดังนั้นจึงจำเป็นที่จะต้องหาวิธีที่การแบ่งคำและการค้นหาความผิดพลาดที่คำนึงถึงปัญหานี้ด้วยเพื่อให้สามารถแก้ปัญหาได้ครอบคลุมมากขึ้น การแบ่งคำสำหรับข้อความที่มีความผิดพลาดนั้นก็ต้องทำโดยคิดว่าภายในข้อความมีความผิดพลาดอยู่และแบ่งคำโดยพยายามรักษาสภาพของส่วนที่ดีเอาไว้ไม่ให้ได้รับผลกระทบจากความผิดพลาดในขณะที่พยายามแบ่งคำโดยพยายามให้ความผิดพลาดปรากฏออกมาเพื่อความสะดวกในการค้นหาความผิดพลาด และการแก้ไขในขั้นต่อๆ ไป

สำหรับคำในภาษาไทยที่มีความผิดพลาดแล้วสิ่งที่ผู้วิจัยสังเกตเห็นอย่างหนึ่งก็คือ เมื่อนำคำนั้นไปผ่านการแบ่งคำก็จะได้คำย่อยๆ ออกมา เช่น "บุเพสันนิวาส" เมื่อผ่านการแบ่งคำจะได้ออกมาเป็น "บุ" , "เพ" และ "สันนิวาส" ซึ่งลักษณะนี้เป็นลักษณะที่พบมากที่สุดเนื่องจากคำในภาษาไทยเป็นคำประสม ซึ่งคือการรวมกันของคำหรือพยางค์ และคำสั้นๆ ก็ไม่ค่อยเหมาะสมสำหรับการทำดัชนีอยู่แล้วจึงเป็นสิ่งที่ควรทำการแก้ไขอย่างมาก สำหรับอีกกรณีหนึ่งคือความผิดพลาดทำให้คำ 2 คำรวมกัน เช่น "เที่ยงธรรม" ซึ่งเกิดจาก "เที่ยว" ซึ่งพิมพ์แทน "ว" ด้วย "ง" ซึ่งจะกลายเป็น "เที่ยง" และ "ธรรม" กรณีนี้ก็เกิดขึ้นพอสมควรและวิธีการที่คิดขึ้นยังไม่สามารถจัดการกับกรณีนี้ได้ แต่ก็เป็นอีกประเด็นที่น่าสนใจสำหรับการพัฒนาขั้นต่อไป

จากที่กล่าวไปแล้วข้างต้นว่าความผิดพลาดมักจะทำให้คำแตกออกเป็นส่วนๆ ดังนั้นการแก้ปัญหาก็ควรจะเริ่มจากทำอย่างไรจึงจะสังเกตได้ว่าคำๆหนึ่งที่ผ่านจากกระบวนการแบ่งคำนั้นเป็นคำที่แตกออกมาจากคำที่เกิดความผิดพลาด ถ้ามองย้อนกลับเราจะเห็นว่าคำที่แตกออกมานั้นเป็นส่วนประกอบของคำที่ผิด กล่าวคือเป็นคำที่มีความสามารถที่จะรวมกันคำอื่นเป็นคำที่ใหญ่กว่าได้ซึ่งส่วนที่แตกออกมานั้นอาจเป็นส่วนย่อยของคำหลายๆ คำในพจนานุกรม กล่าวคือสามารถรวมกับคำอื่นๆ หลายคำเพื่อทำให้เกิดคำใหม่หรืออาจพูดว่ามีความสามารถในการรวมสูง ตัวอย่างเช่น "กระรอก" แต่เขียนผิดเป็น "กระอก" เมื่อผ่านการตัดคำจะถูกแยกเป็น "กระ" และ "อก" จะเห็นได้ว่าทั้ง "กระ" และ "อก" เป็นสตริงที่เป็นส่วนหนึ่งของคำหลายคำในพจนานุกรม เช่น "กระ" เป็นส่วนหนึ่งของ "กระรอก" , "กระบอก" , "กระจาย" , "กระบวน" , "กระบวย" ฯลฯ ส่วน "อก" เป็นส่วนหนึ่งของ "กระรอก" , "กระบอก" , "คอก" , "จอก" , "ดอก" , "ลูกดอก" ฯลฯ แต่เมื่อพิจารณาทั้งสองส่วนแล้วจะเห็นได้ว่า "กระอก" นั้นน่าจะเกิดจากการพิมพ์ตกโดยอาจเป็นคำ "กระรอก" หรือ "กระบอก" ดังนั้นค่านี้จึงเป็นค่าหนึ่งที่น่าจะใช้เป็นเครื่องช่วย

ในการจับความผิดพลาด ซึ่งผู้วิจัยขอนิยามค่าดังกล่าวเป็นค่า "ความเฉพาะตัว" $U(s)$ ของสตริง $s$ ใหม่ดังนี้

$U(s)$ = จำนวนครั้งของ $s$ ที่ปรากฏเป็นส่วนหนึ่งของคำในพจนานุกรม

ตัวอย่าง $U$ ("นิวาส" ) = 4

เพราะ "นิวาส" ปรากฏใน "นิวาส" , "บุพเพนิวาสานุสติญาณ" , "บุพเพสันนิวาส" และ "สันนิวาส"

### 4.1 ที่มาของค่าความเฉพาะตัว

ค่าความเฉพาะตัวของคำหรือสตริงนั้นคือจำนวนครั้งที่คำหรือสตริงนั้นๆปรากฏในพจนานุกรม โดยอาจเป็นเพียงส่วนหนึ่งของคำก็ได้ ซึ่งผู้วิจัยได้แนวความคิดของค่านี้มาจากตาราง n-gram แบบที่เก็บข้อมูลเป็นความถี่อย่างไรก็ดีการใช้ ตาราง n-gram นั้นมีข้อจำกัดคือจะใช้ได้กับสตริงที่มีความยาวเท่ากับ ค่า n ที่ใช้ในการสร้างตารางการใช้ค่า n ที่มากก็จะสูญเสียพื้นที่ในหน่วยความจำมาก ผู้วิจัยได้ทดลองปรับปรุง ตาราง n-gram ให้สามารถหาค่านี้สำหรับทุกความยาวตัวอักษรโดยใช้ค่า n เป็น 2 ในการสร้าง(ใช้เนื้อที่ในหน่วยความจำประมาณ 2.5 MB)แต่เมื่อนำมาใช้จริงเกิดปัญหาเนื่องจากทำให้การประมวลผลช้ามากจึงต้องนำแนวความคิดนี้ไปประยุกต์กับโครงสร้างข้อมูลแบบทรัยดังจะกล่าวต่อไปแทน

### 4.2 ความสำคัญของค่าความเฉพาะตัว

ค่านี้นอกจากสามารถบอกว่า สตริงใดๆ เป็นสตริงที่ถูกต้องหรือไม่เมื่อเทียบกับพจนานุกรม ในกรณีที่ค่าเป็น 0 แสดงถึงสตริงนั้นเป็นสตริงที่ไม่ถูกต้อง ส่วนกรณีที่ค่ามากกว่า 0 แสดงถึงถึงสตริงนั้นเป็นสตริงที่ถูกต้อง และยังบอกว่าสตริงนั้นเกิดขึ้นมากเพียงไร การที่เกิดมากหมายถึงสตริงนั้นเป็นส่วนประกอบของคำหลายคำในพจนานุกรม ถ้ามองอีกมุมหนึ่งก็คือสตริงนี้มีศักยภาพสูงในการที่จะรวมกับตัวอักษรข้างเคียงและทำให้เกิดคำที่ถูกต้องในพจนานุกรม ดังที่กล่าวไปแล้วว่าความผิดพลาดมักจะทำให้คำแยกออกเมื่อนำคำไปผ่านกระบวนการแบ่งคำ ดังนั้นค่านี้จึงเป็นค่าที่เหมาะสมที่จะใช้ในการค้นหาความผิดพลาด ประโยชน์อีกอย่างหนึ่งของค่านี้ก็คือสามารถนำไปประยุกต์ใช้ในการทำดัชนีได้ กล่าวคือคำที่มีค่าความเฉพาะตัวต่ำเป็นคำที่เหมาะสมสำหรับการทำดัชนีอย่างไรก็ดีการใช้ค่านี้ต้องใช้ร่วมกับความยาวของสตริงเพราะเมื่อจำนวนตัวอักษรน้อยจะส่งผลให้ค่านี้สูงอยู่แล้ว นอกจากนี้ผู้วิจัยยังได้นำค่านี้ไปใช้ในการแบ่งคำด้วยเพื่อให้การแบ่งคำมีความเหมาะสมสำหรับการทำดัชนี

### 4.3 การประยุกต์โครงสร้างข้อมูลแบบทรัยเพื่อหาค่าความเฉพาะตัวของคำ

โดยลักษณะของโครงสร้างข้อมูลแบบทรัย ซึ่งสนับสนุนการค้นหาอักขระเติมหน้า (prefix) ไม่สามารถรองรับการหาสตริงซึ่งอยู่กลางคำได้ดังนั้นจึงต้องมีการปรับปรุง อีกจุดหนึ่งที่ต้องปรับก็คือการเพิ่มค่าความเฉพาะตัวเข้าไปเพื่อความรวดเร็วในการหา

การปรับปรุงเพื่อให้สามารถหาสตริงซึ่งอยู่กลางคำได้ สามารถทำได้โดยการใส่อักขระเติมหลัง (suffix) ทุกส่วนของคำเข้าไปด้วย เช่น เมื่อจะใส่คำว่า "กระรอก" เข้าไปในทรัยก็ให้ใส่ "กระรอก" รวมถึง "ระรอก" , "ะรอก" , "รอก" , "อก" และ "ก" เข้าไปในทรัยด้วย

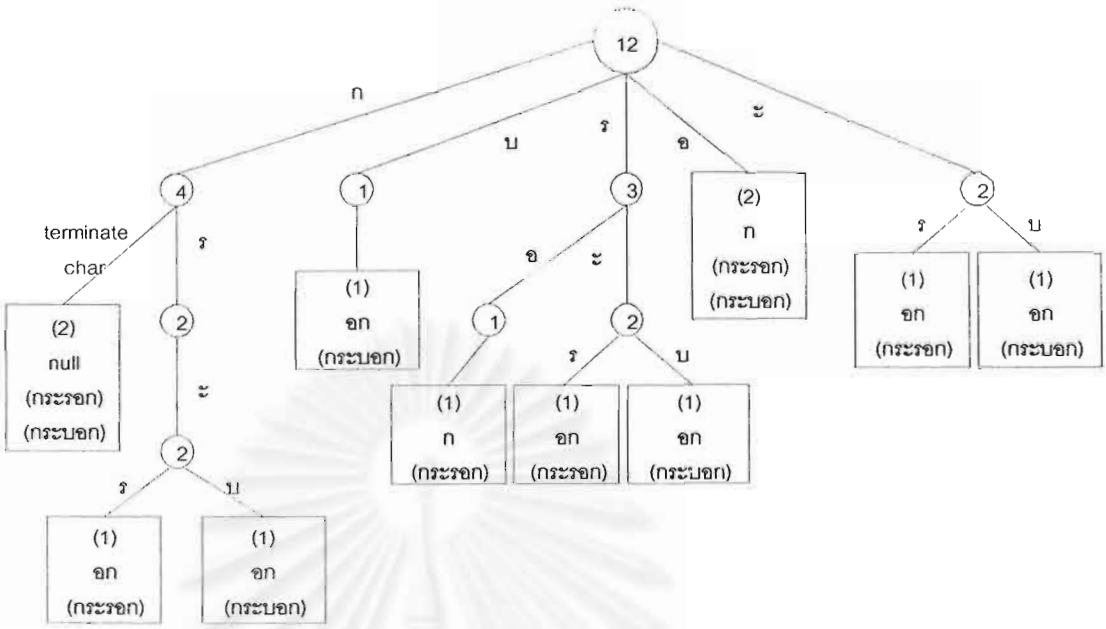
การเพิ่มค่าความเฉพาะตัวนั้นสามารถทำได้ไม่อยากเพียงแค่เพิ่มตัวนับเข้าไป ดังแสดงในรูปที่ 4.1 โดยคำที่ใส่เข้าไปในทรัยมีเพียง "กระรอก" และ "กระบอก"

### 4.4 การแบ่งคำภาษาไทย

ขั้นตอนการแบ่งคำภาษาไทยนี้พัฒนาขึ้นมาเองเนื่องจากโปรแกรมการแบ่งคำที่มีอยู่โดยส่วนใหญ่เป็นโปรแกรมแบ่งพยางค์ที่ใช้สำหรับตัดคำเพื่อขึ้นบรรทัดใหม่ในโปรแกรมประมวลผลคำ และโปรแกรมแบ่งคำเหล่านั้นทำโดยไม่คำนึงถึงว่าประโยคที่นำมาแบ่งคำนั้นมีความผิดพลาดปนอยู่หรือไม่ โปรแกรมการแบ่งคำที่พัฒนาขึ้นนี้เป็นโปรแกรมแบ่งคำโดยใช้พจนานุกรมซึ่งพัฒนาขึ้นเพื่อให้ได้ผลลัพธ์ที่เหมาะสมสำหรับการทำดัชนีและพยายามแบ่งคำโดยให้ผลกระทบต่อคำข้างเคียงของคำที่มีความผิดพลาดเกิดขึ้นน้อย ขั้นตอนในการแบ่งคำภาษาไทยที่นำเสนอในที่นี้จะประกอบด้วยขั้นตอนย่อยๆเป็นขั้นๆ ซึ่งจะอธิบายโดยใช้ตัวอย่างที่ 1 ประกอบซึ่งในตัวอย่างนี้ทำโดยสมมติว่า "หน่วยความจำ" เป็นคำที่ปรากฏในพจนานุกรม

ตัวอย่างที่ 1 "การผลิตหน่วยความจำเป็นจำนวนมาก"

ขั้นตอนที่ 1 ดึงคำที่ยาวที่สุดออกมาจากข้อความ โดยที่คำที่ดึงออกมานี้ต้องไม่เป็นส่วนหนึ่งของคำอื่นโดยสมบูรณ์ แต่สามารถให้เหลื่อมหรือทับกันเพียงบางส่วนได้ จากตัวอย่างที่ 1 จะได้คำ "การ" , "ผลิต" , "หน่วยความจำ" , "จำเป็น" , "จำนวน" , "นม" และ "มาก" ข้อสังเกตจะไม่ได้คำว่า "ความ" หรือ "จำ" ออกมาเพราะเป็นส่วนหนึ่งของคำว่า "หน่วยความจำ" โดยสมบูรณ์ ในขณะที่จะได้ "นม" ออกมาเนื่องจาก "นม" ไม่ได้เป็นส่วนหนึ่งของ "จำนวน" หรือ "มาก" โดยสมบูรณ์เพียงแค่เหลื่อมกันเท่านั้น ขั้นตอนนี้เขียนเป็นรหัสเทียมได้รูปที่ 4.2 โดยที่ $T$ ในรหัสเทียมก็คือข้อความ "การผลิตหน่วยความจำเป็นจำนวนมาก" และ $f_{i,j}$ ก็คือคำยาวที่สุด ณ ตำแหน่งต่างๆที่ดึงออกมาจากข้อความ เช่น $f_{0,2}$ ก็คือ "การ" , $f_{3,6}$ ก็คือ "ผลิต"




**รูปที่ 4.1 แสดงโครงสร้างข้อมูลแบบทรัยที่ถูกประยุกต์สำหรับการหาค่าความเฉพาะตัว**

```
T (t0t1t2 … tl-1) = input_text;
base_set = { }
for each position i in text T (t0t1t2 … tl-1)
{
    find longest word fi,j start from position i that exist as
    a word in dictionary where i is start position of word in
    T and j is stop position of word in T
    if (fi,j not completely inside any word in base_set)
    {
            add fi,j to base_set
     }
}
```

**รูปที่ 4.2 แสดงรหัสเทียมสำหรับการแบ่งคำในขั้นตอนที่ 1**

ขั้นตอนที่ 2 สร้างคำเพิ่มเติม ในกรณีที่คำที่ได้จากขั้นตอนที่ 1 เหลื่อมกัน( $f_{i,j}$ เหลื่อมกับ $f_{m,n}$ ถ้า $j > m$) การสร้างคำเพิ่มจากขั้นตอนนี้มี 2 แบบ คือการแบ่งออกเป็น 3 ส่วน ซึ่งจะแบ่ง $f_{i,j}$ ที่เหลื่อมกับ $f_{m,n}$ ออกเป็น 3 ส่วนดังนี้ $f_{i,m-1}$ , $f_{m,j}$ และ $f_{j+1,n}$ ส่วนการแบ่งเป็น 2 ส่วนจะแบ่ง $f_{i,j}$ ที่เหลื่อมกับ $f_{m,n}$ ออกเป็น 2 ส่วนถ้าจำนวนตัวอักษรที่เหลื่อมกันมีมากกว่า 2 ($j > m$) โดยสำหรับทุกตำแหน่ง $j \le k < m$ จะแบ่ง $f_{i,j}$ และ $f_{m,n}$ ออกเป็น $f_{i,k}$ และ $f_{k+1,n}$ จากตัวอย่างจะได้คำ "หน่วยความ" , "จำ" และ "เป็น" ออกมา จากการแบ่งออกเป็น 3 ส่วน และ "หน่วยความจ"

และ "-ำเป็น" จากการแบ่งออกเป็น 2 ส่วนของคำ "หน่วยความจำ" และ "จำเป็น" ที่เหลื่อมกันที่ได้จากขั้นตอนที่ 1 และได้คำ "จำนว" , "น" และ "ม" จากการแบ่งเป็น 3 ส่วน ของคำ "จำนวน" และ "นม" ที่เหลื่อมกันที่ได้จากขั้นตอนที่ 1 ข้อสังเกตจะไม่มีคำที่ได้จากการแบ่งเป็น 2 ส่วนสำหรับกรณีนี้เนื่องจากมีการเหลื่อมกันเพียงตัวอักษรเดียว และได้คำ "น" , "ม" และ "าก" จากการแบ่งเป็น 3 ส่วน ของคำ "นม" และ "มาก" ที่เหลื่อมกันที่ได้จากขั้นตอนที่ 1 ขั้นตอนนี้เขียนเป็นรหัสเทียม ได้ดังรูปที่ 4.3

```
extend_set = {}
for each f_{i,j}  in base set
{
  for each f_{m,n}  in base set
  {
    If  f_{i,j}  overlap with f_{m,n}
    {
      if  (j>m)
      {
        // tree pieces separate
        add f_{i,m-1} to extend_set
        add f_{m,j} to extend_set
        add f_{j+1,n}to extend_set
      }

      for each position k  j ≤ k < m
      {
        // two pieces separate
        add f_{i,k} to extend_set
        add f_{k+1,n} to extend_set
      }
    }
  }
}
```

**รูปที่ 4.3 แสดงรหัสเทียมสำหรับการแบ่งคำในขั้นตอนที่ 2**

ขั้นตอนที่ 3 สร้างคำเพิ่มเติมจากคำที่ได้จากขั้นตอนที่ 1 ในกรณีที่ คำที่ได้จากขั้นตอนที่ 1 มีส่วนของคำเป็นคำที่อยู่ในพจนานุกรม จากตัวอย่างที่ 1 จะได้คำ "หน่วย" , "ความ" , "วาม" และ "จำ"(ตำแหน่งแรก) เนื่องจากคำเหล่านี้เป็นคำที่ปรากฏอยู่ในพจนานุกรมและเป็นส่วนหนึ่งของคำว่า "หน่วยความจำ" ที่ได้จากขั้นตอนที่ 1 และ "จำ"(ตำแหน่งที่สอง) และ "นว" จากคำว่า "จำนวน" ที่ได้จากขั้นตอนที่ 1 ขั้นตอนนี้เขียนเป็นรหัสเทียมได้ดังรูปที่ 4.4

ขั้นตอนที่ 4 กำจัดคำซ้ำซ้อนที่เกิดขึ้น จากตัวอย่างที่ 1 จะได้คำทั้งหมดหลังจากกำจัดความซ้ำซ้อนคือ "การ" , "ผลิต" , "หน่วยความจำ" , "จำเป็น" , "จำนวน" , "นม" , "มาก" , "หน่วยความ" , "จำ""(ตำแหน่งแรก) , "เป็น" , "หน่วยความจ" , "-ำเป็น" , "จำนว" , "น" , "ม" , "าก" , "ความ" , "วาม" , "จำ"(ตำแหน่งที่สอง) และ "นว"

```
for each f_{i,j}  in base set
{
  if exist f_{i,j}  that f_{m,n}  is a word exist in dictionary
      where i <= m and j >= n and f_{i,j}  ! = f_{m,n}
  {
    add f_{m,n}   to extend_set
   }
 }
```

**รูปที่ 4.4 แสดงรหัสเทียมสำหรับการแบ่งคำในขั้นตอนที่ 3**

ขั้นตอนที่ 5 ให้น้ำหนักคำสำหรับแต่ละคำ และสร้างกราฟโดยที่คำจะเป็นจุด (vertice) และสร้างเส้น(edge) เชื่อมระหว่างคำที่ติดกันโดยที่น้ำหนักของเส้นจะมีค่าเท่ากับน้ำหนักของคำปลายทาง สำหรับการให้น้ำหนักมีกฎดังนี้

สำหรับสตริง $f_{i,j}$ $(t_i t_{i+1} t_{i+2} \dots t_j)$ จากข้อความ $T(t_0 t_1 t_2 \dots t_{L-1})$ โดยที่ $i,j$ $[0,L-1]$ จะให้น้ำหนัก $W(f_{i,j})$ สำหรับสตริง $f_{i,j}$ ดังนี้ (โดยที่ $W(f_{i,j})$ ยิ่งน้อยแสดงถึง $f_{i,j}$ เป็นสตริงที่ดี)

โดยที่

$\beta$ เป็นจำนวนเต็มในช่วง [2-20] ใช้เป็นตัวปรับการให้ความสำคัญของความยาวเป็นตัวอักษรของ $f_{i,j}$

$\gamma$ เป็นจำนวนเต็มในช่วง [1-10] ซึ่งใช้เป็นตัวปรับค่า $W(f_{i,j})$ ของ $f_{i,j}$ ที่เป็นสตริงที่ไม่ถูกต้องเช่น “ขขข” มีค่ามากกว่า $W(f_{m,n})$ ของ $f_{m,n}$ ที่เป็นสตริงที่ถูกต้อง เช่น “กระจา” ซึ่งเป็นสตริงที่ถูกต้องถึงแม้จะไม่เป็นคำที่ปรากฏในพจนานุกรม แต่ “กระจา”เป็นส่วนหนึ่งของคำ “กระจาด” , “กระจาย” , “กระจัดกระจาย” ซึ่งปรากฏในพจนานุกรม

$\delta = \beta b$ โดยที่ $b = W(s)$ โดยที่ $s$ เป็นสตริงที่ปรากฏบ่อยที่สุดในพจนานุกรม

ค่า $W(f_{i,j})$ ของ $f_{ij}$ คำนวณได้จากสูตรดังนี้

$W(f_{i,j}) = \lfloor \beta W(f_{i,j}) / (j-i+1) \rfloor$ ถ้า ความยาวเป็นตัวอักษร$(f_{i,j}) \geq 2$ และ $U(f_{i,j}) \geq 1$ และ $f_{i,j}$ เป็นคำที่ปรากฏในพจนานุกรม

$W(f_{i,j}) = \lfloor \delta + \beta W(f_{i,j}) / (j-i+1) \rfloor$ ถ้า ความยาวเป็นตัวอักษร$(f_{i,j}) \geq 2$ และ $U(f_{i,j}) \geq 1$ และ $f_{i,j}$ ไม่ปรากฏพจนานุกรม

$W(f_{i,j}) = \lfloor \delta + \beta \rfloor$ ถ้า ความยาวเป็นตัวอักษร$(f_{i,j}) = 1$

หอสมุดกลาง สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย



$W(f_{i,j}) = \lfloor \gamma(2\delta) \rfloor$ ถ้า ความยาวเป็นตัวอักษร$(f_{i,j}) \geq 2$ และ

$U(f_{i,j}) = 0$

จากวิธีการให้น้ำหนักดังกล่าวจะสามารถแยกสตริงออกเป็น 4 กลุ่มคือ

<u>กลุ่มที่ 1</u> สตริง $f_{i,j}$ เป็นคำที่ปรากฏในพจนานุกรมกลุ่มนี้จะมีค่า $W(f_{i,j}) = \beta W(f_{i,j})/(j\text{-}i\text{+}1)$ข้อสังเกตค่า $W(f_{i,j})$ ของกลุ่มนี้จะมีค่าต่ำสุด ตัวอย่างคำในกลุ่มนี้เช่น "กระจาย"

<u>กลุ่มที่ 2</u> สตริง $f_{i,j}$ เป็นสตริงที่ถูกต้องแต่ไม่ใช่คำที่ปรากฏในพจนานุกรม กลุ่มนี้จะมีค่า $W(f_{i,j}) = \delta+\beta W(f_{i,j})/(j\text{-}i\text{+}1)$ ซึ่งค่า $W(f_{i,j})$ ของกลุ่มนี้จะมีค่ามากกว่าค่า $W(f_{i,j})$ ของกลุ่มที่ 1 ซึ่งเป็นผลมาจากค่า $\delta$ ตัวอย่างสตริงในกลุ่มนี้ เช่น "กระจา" ซึ่งเป็นสตริงที่ถูกต้องถึงแม้จะไม่เป็นคำที่ปรากฏในพจนานุกรม แต่ "กระจา"เป็นส่วนหนึ่งของคำ "กระจาด" , "กระจาย" , "กระจัดกระจาย" ซึ่งปรากฏในพจนานุกรม

<u>กลุ่มที่ 3</u> $f_{i,j}$ เป็นตัวอักษรเดี่ยว กลุ่มนี้จะมีค่า $W(f_{i,j}) = \delta+\beta$ ซึ่งค่า $W(f_{i,j})$ ของกลุ่มนี้จะน้อยกว่าสตริงที่จัดอยู่ในกลุ่มที่ 2 สาเหตุที่ให้ค่าของกลุ่มนี้น้อยกว่ากลุ่มที่ 2 เนื่องจากผู้วิจัยพบว่าในกรณีความผิดพลาดที่เกิดจากการพิมพ์เกิน ตัวอักษรที่เกินมานั้นมีโอกาสสูงที่จะรวมกับตัวอักษรใกล้เคียงและเกิดเป็นคำที่มีในพจนานุกรม และส่งผลกระทบทำให้การตัดคำได้ผลลัพธ์ที่ไม่ดีถ้าเราให้ค่า $W(f_{i,j})$ ของกลุ่มนี้สูง เช่น "ประกอบกกระทง" จะได้ผลลัพธ์ของการตัดคำเป็น "ประกอ" , "บก" , "กระทง" ถ้าเราให้ค่า $W(f_{i,j})$ ของกลุ่มนี้สูงกว่ากลุ่มที่ 2 ในขณะที่จะได้ผลลัพธ์ของการตัดคำเป็น "ประกอบ" , "ก" , "กระทง" ถ้าเราให้ค่า $W(f_{i,j})$ ของกลุ่มนี้ต่ำกว่ากลุ่มที่ 2 ซึ่งจะเห็นว่าสามารถแยกตัวอักษรที่เกินออกมาได้

<u>กลุ่มที่ 4</u> $f_{i,j}$ เป็นสตริงที่ไม่ถูกต้อง กลุ่มนี้จะมีค่า $W(f_{i,j})$มากที่สุดคือ $W(f_{i,j}) = \gamma(2\delta)$ซึ่งค่า $W(f_{i,j})$ ของกลุ่มนี้จะมากกว่าค่า $W(f_{i,j})$ ของทุกกลุ่ม ตัวอย่างสตริงในกลุ่มนี้เช่น "ขขข"

จากตัวอย่างที่ 1 จะได้กราฟดังรูปที่ 4.5 (dummy ใช้เพื่อให้การหาเส้นทางสั้นสุดทำได้ง่ายเท่านั้น)

<u>ขั้นตอนที่ 6</u> หาเส้นทางสั้นสุด และ ใช้เส้นทางสั้นสุดในการตัดคำ ด้วยวิธีการให้น้ำหนักสตริงที่กล่าวข้างต้นจะทำมีสตริงในกลุ่มที่ 2 อยู่ในเส้นทางสั้นสุดมากที่สุดเนื่องจากมีน้ำหนักน้อยที่สุด และตามด้วยสตริงในกลุ่มที่ 1 , 3 และ 4 ตามลำดับ จากตัวอย่างจะได้เส้นทางสั้นสุดตามเส้นหนาดังแสดงในรูปที่ 4.5 และจะได้ผลการแบ่งคำคือ "การ" , "ผลิต" , "หน่วยความจำ" , "เป็น" , "จำนวน" , "มาก"

ตัวอย่างที่ 2 "กับแกบล้มไม่สะอด"

ขั้นตอนที่ 1 จะได้ "กับ" , "แก" , "กบ" , "ล้ม" , "ไม่" , "สะ" และ "อด" จากการคำที่ยาวที่สุดออกมาจากข้อความ

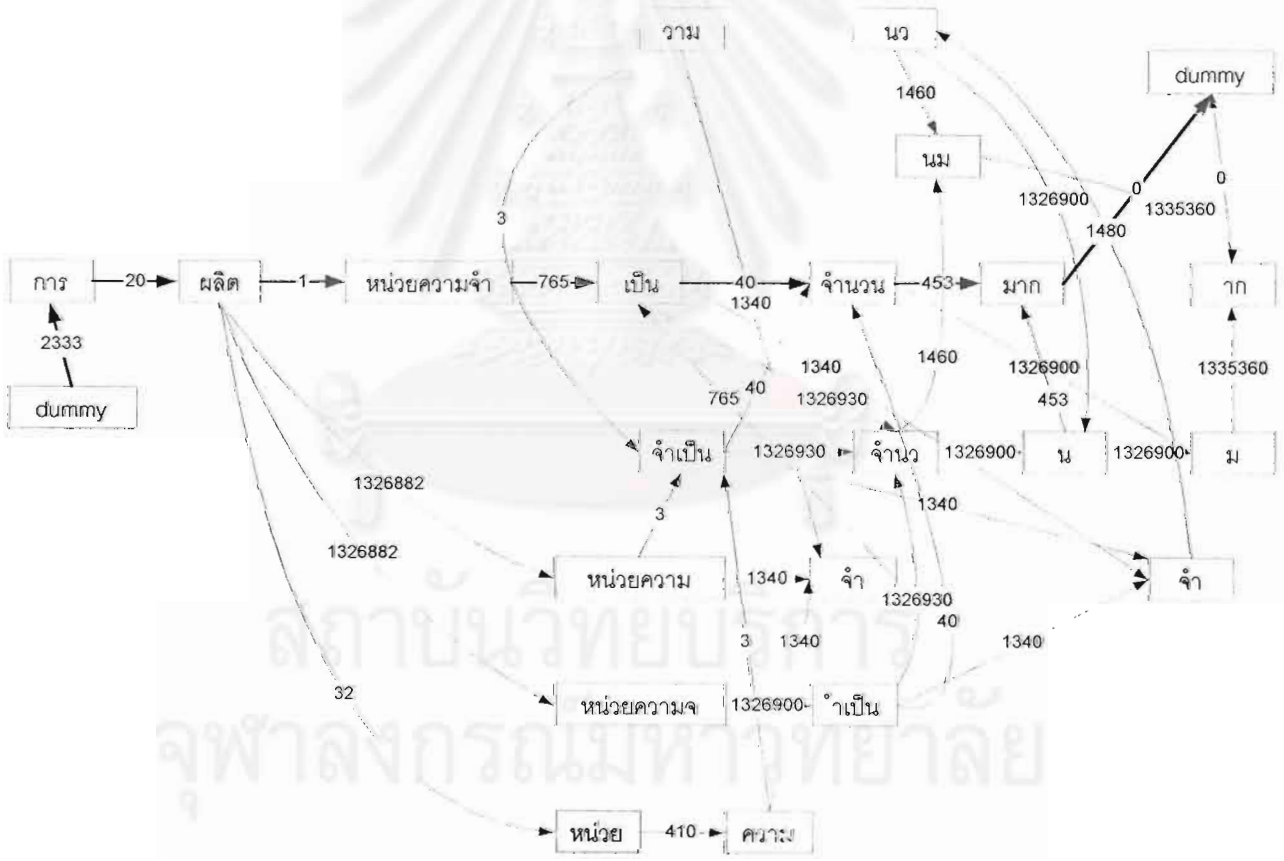
ขั้นตอนที่ 2 จะได้ "แ" , "ก" , "บ" จากการแบ่ง "แก" และ "กบ" ออกเป็น 3 ส่วน

ขั้นตอนที่ 3 จะไม่ได้คำเพิ่มเติมจากขั้นตอนที่ 3 สำหรับตัวอย่างที่ 2

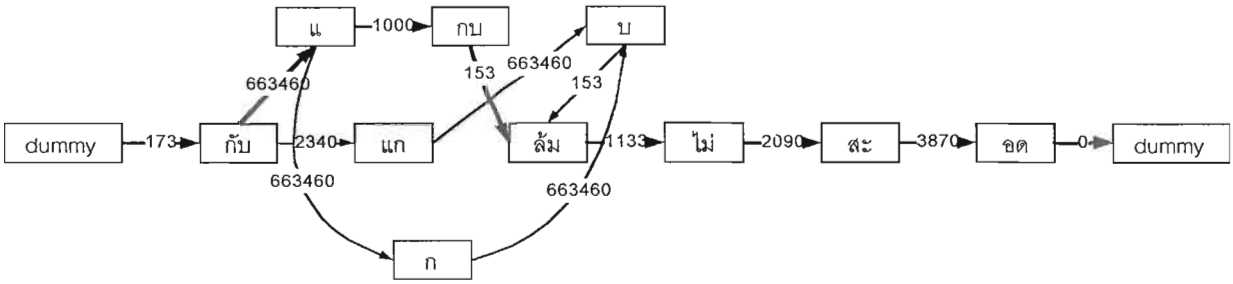
ขั้นตอนที่ 4 ไม่มีคำซ้ำซ้อนที่ต้องกำจัดสำหรับตัวอย่างที่ 2 ดังนั้นคำทั้งหมดที่ได้ ณ จุดนี้คือ "กับ" , "แก" , "กบ" , "ล้ม" , "ไม่" , "สะ" , "อด" , "แ" , "ก" และ "บ"

ขั้นตอนที่ 5 เมื่อนำไปสร้างกราฟจะได้ดังรูปที่ 4.6

ขั้นตอนที่ 6 จะได้เส้นทางสั้นสุดดังแสดงในรูปที่ 4.6 และได้ผลลัพธ์การตัดคำคือ "กับ" , "แ", "กบ" , "ล้ม" , "ไม่" , "สะ" , "อด"




**รูปที่ 4.5 แสดงกราฟที่ได้จากกระบวนการแบ่งคำในขั้นตอนที่ 5,6 ของตัวอย่างที่ 1**




**รูปที่ 4.6 แสดงกราฟที่ได้จากกระบวนการแบ่งคำในขั้นตอนที่ 5,6 ของตัวอย่างที่ 2**

### 4.5 การค้นหาความผิดพลาดในข้อความภาษาไทย

หลังจากที่ประโยคถูกแบ่งเป็นคำแล้ว ถ้าค่า $W(f_{i,j})$ ของ $f_{i,j}$ ใดๆที่มีค่ามากกว่าค่าความเป็นไปได้ของความผิดพลาด $E(f_{i,j})$ ของ $f_{i,j}$ ก็ให้สงสัยว่าส่วนนั้นอาจเกิดจากความผิดพลาดเพื่อใช้ในการแก้ไขความผิดพลาดในขั้นต่อไป การคำนวณค่า $E(f_{i,j})$ สามารถทำได้ดังนี้

$$E(f_{i,j}) = \left\lfloor \frac{\lambda}{length(f_{i,j})^2} \right\rfloor$$

โดยที่ $\lambda$ เป็นค่าคงที่ที่สามารถปรับได้อยู่ช่วง [10β,2000β]

$length(f_{i,j})$ เป็นความยาวเป็นตัวอักษรของ $f_{i,j}$

ค่า $\lambda$ จะเป็นตัวกำหนดว่าสตริง $f_{i,j}$ ใดน่าสงสัยสำหรับสตริงในกลุ่มที่ 1 สำหรับกลุ่มอื่นๆ ค่า $W(f_{i,j}) > E(f_{i,j})$ แน่นอนซึ่งจะถูกจำไว้ว่าเป็นความผิดพลาดและใช้ในการแก้ไขความผิดพลาดในขั้นต่อไป ส่วนสตริง $f_{i,j}$ ในกลุ่มที่ 1 นั้นสามารถทำให้ความสัมพันธ์ระหว่างค่า $W(f_{i,j})$ และ $E(f_{i,j})$ เป็นได้ทั้งสองรูปแบบคือ $W(f_{i,j}) > E(f_{i,j})$ และ $W(f_{i,j}) \le E(f_{i,j})$ กล่าวคือการที่ส่วนสตริง $f_{i,j}$ ในกลุ่มที่ 1 จะถูกวินิจฉัยว่าเป็นความผิดพลาดหรือไม่นั้นขึ้นอยู่กับการตั้งค่า $\lambda$ ค่า $\lambda$ จะมีผลต่อการประมวลผลดังนี้คือ ค่า $\lambda$ ที่น้อยจะทำให้ได้ส่วนที่สงสัยว่าจะเป็นความผิดพลาดมากซึ่งบางครั้งส่วนนั้นอาจไม่ใช่ความผิดพลาดจริงและทำให้การประมวลผลในส่วนการแก้ไขความผิดพลาดช้า ในขณะที่ถ้าค่า $\lambda$ ที่มากก็อาจทำให้ค้นหาความผิดพลาดไม่พบแต่จะสามารถทำการประมวลผลในส่วนของการแก้ไขความผิดพลาดได้เร็ว ตารางที่ 4.1 แสดงค่าความเฉพาะตัวของ $f_{i,j}$ ที่ทำให้ค่า $W(f_{i,j}) > E(f_{i,j})$ เมื่อเทียบกับความยาวเป็นตัวอักษรของ $f_{i,j}$ เมื่อค่า $\lambda = 100\beta$ ตัวอย่างเช่น “สิ” ความยาว 2 ตัวอักษรมีค่าความเฉพาะตัว 78 เมื่อเทียบกับตารางที่ 4.1 จะพบว่า 78 > 50 ดังนั้นคำว่า “สิ” จะถูกพิจารณาว่าเป็นความผิดพลาด ถ้าพิจารณาความสัมพันธ์ระหว่างค่า $W(f_{i,j})$ และ $E(f_{i,j})$ เมื่อ $W(f_{i,j}) > E(f_{i,j})$ จะได้ความสัมพันธ์ระหว่าง $\lambda$ กับค่าความ

เฉพาะตัวของ $f_{i,j}$ หรือ $U(f_{i,j})$ ในอสมการ ดังนี้(พิจารณา $\lambda = x\beta$ โดยที่ $x$ เป็นค่าคงที่อยู่ในช่วง [10,2000])

$$E(f_{i,j}) = \frac{\lambda}{length(f_{i,j})^2} < \frac{\beta U(f_{i,j})}{length(f_{i,j})} = W(f_{i,j})$$

$$\frac{\lambda}{\beta(length(f_{i,j})} < U(f_{i,j})$$

**ตารางที่ 4.1 แสดงค่า$U(f_{i,j})$ ที่ทำให้ค่า $W(f_{i,j}) > E(f_{i,j})$ เมื่อเทียบกับความยาวเป็นตัวอักษรของ $f_{i,j}$ เมื่อค่า $\lambda = 100\beta$**

| ความยาวเป็นตัวอักษรของ $f_{i,j}$ | ค่า $U(f_{i,j})$ ที่ทำให้ค่า $W(f_{i,j}) > E(f_{i,j})$ |
|---|---|
| 2 | >50 |
| 3 | >33 |
| 4 | >25 |
| 5 | >20 |
| 6 | >16 |
| 7 | >14 |
| 8 | >12 |
| 9 | >11 |
| 10 | >10 |

ตัวอย่างที่ 3 “สิงโตทำลานกรงออกมา” ประโยคที่ถูกต้องคือ “สิงโตทำลายกรงออกมา” ถ้าแบ่งคำด้วยกระบวนการที่กล่าวข้างต้นและค้นหาความผิดพลาด โดยใช้ค่า $\lambda = 300$ จะได้

“สิ*” , “งง” , “โต*” , “ทำ*” , “ลาน*” , “กรง” , “ออก*” , “มา*”

สตริงที่มี * กำกับคือสตริงที่น่าสงสัยว่าเกิดจากความผิดพลาดกล่าวคือ $W(f_{i,j}) > E(f_{i,j})$ สังเกต “สิ*” และ “โต*” เกิดจากความผิดพลาดซึ่งทำให้คำ “สิงโต” แตกออกเมื่อผ่านกระบวนการแบ่งคำ “ทำ*” และ “ลาน*” เกิดเกิดจากความผิดพลาดซึ่งทำให้คำ “ทำลาย” แตกออกเมื่อผ่านกระบวนการแบ่งคำ ในขณะที่ “ออก*” และ “มา*” ไม่ได้เกิดจากความผิดพลาดแต่อย่างใดแต่เนื่องจากค่า $W(f_{i,j})$ ของคำทั้งสองสูงมาก เนื่องจากคำทั้งสองเป็นสตริงที่เกิดขึ้นมากในพจนานุกรม และส่งผลให้ค่า $W(f_{i,j}) > E(f_{i,j})$ข้อสังเกตบางส่วนของคำผิดที่แตกออกมาอาจจะไม่

แสดงออกโดยมี * กำกับเช่น “งง” วิธีนี้ขอให้มีส่วนหนึ่งของคำผิดที่แตกออกมามี * กำกับเพียงส่วนเดียวก็จะสามารถหาทางแก้ไขได้เช่น “สิงโต” พิมพ์ผิดเป็น “สิงงโต” เมื่อผ่านการค้นหาความผิดพลาดจะได้ “สิ*” “งง” “โต*” ซึ่งมี * ถึงสองตำแหน่งซึ่งเพียงพอสำหรับการแก้ไขในขั้นตอนต่อไป

### 4.6 การแก้ไขความผิดพลาดในข้อความภาษาไทย

หลังจากได้ข้อความที่แบ่งแล้วและข้อมูลของส่วนที่น่าสงสัยว่าเกิดจากความผิดพลาด การแก้ไขความผิดพลาดเพื่อหาคำที่จะเพิ่มเข้าไปในดัชนีเพื่อให้ได้ดัชนีที่มีความสมบูรณ์ขึ้น ทำโดยพิจารณาแต่ละส่วนที่สงสัยว่ามีความผิดพลาด ซึ่งมีขั้นตอนดังนี้

ขั้นตอนที่ 1 รวมคำที่สงสัยว่าเกิดจากความผิดพลาดกับคำข้างเคียง ไปขั้นตอนที่ 2

ขั้นตอนที่ 2 หาค่าความเฉพาะตัวของคำ ถ้า ค่าความเฉพาะตัวเป็น 0 ให้ ไปขั้นตอนที่ 3 แต่ถ้าได้ค่ามากกว่า 0 ให้นำคำที่ได้จากการรวม(ให้คิดว่าเป็นคำที่สงสัยว่าเกิดจากความผิดพลาด)กลับไปประมวลผลในขั้นตอนที่ 1 อีกครั้ง

ขั้นตอนที่ 3 ทำการจับคู่แบบประมาณ เก็บคำที่ได้จากการจับคู่แบบประมาณทั้งหมดไปใช้ในการประมวลผลต่อในขั้นตอนที่ 4 โดยประมวลผลทีละคำ

ขั้นตอนที่ 4 นำคำที่ได้รวมกับคำข้างเคียง ไปขั้นตอนที่ 5

ขั้นตอนที่ 5 หาค่าความเฉพาะตัวของคำ ถ้า ค่าความเฉพาะตัวเป็น 0 ให้ ตรวจสอบว่าคำก่อนที่จะรวมเป็นคำในพจนานุกรมหรือไม่ ถ้าเป็นคำในพจนานุกรมให้เก็บไว้ ถ้าไม่ใช่คำในพจนานุกรมให้ทิ้งไป ถ้า ค่าความเฉพาะตัวมากกว่า 0 ให้นำคำที่ได้จากการรวมกลับไปประมวลผลในขั้นที่ 4 อีกครั้ง

คำทั้งหมดที่ได้จากขั้นตอนที่ 5 คือคำที่ควรเพิ่มเข้าไปในดัชนี

เพื่อความเข้าใจการทำงานในขั้นตอนการแก้ไขความผิดพลาดในข้อความภาษาไทย พิจารณาตัวอย่างที่ 3 ประกอบ

ขั้นตอนที่ 1 รวม “สิ” กับ “งง” เป็น “สิงง”

ขั้นตอนที่ 2 คำนวณค่าความเฉพาะตัวได้ค่าเป็น 0 ไปขั้นตอนที่ 3

ขั้นตอนที่ 3 ทำการทำการจับคู่แบบประมาณ ได้ คำ “สิง”(ตัด “ง” ทิ้ง) , “สิ่ง” (เปลี่ยน “ง” เป็น ไม้เอก) , “สิงห”(เปลี่ยน “ง” เป็น “ห”)

ขั้นตอนที่ 4 นำแต่ละคำรวมกับ “โต” เป็น “สิงโต” , “สิ่งโต” และ “สิงหโต”

ขั้นตอนที่ 5

สำหรับ “สิ่งโต” คำนวณค่าความเฉพาะตัวได้ค่าเป็น 0 แต่ “สิ่ง” เป็นคำในพจนานุกรมจึงเก็บคำว่า “สิ่ง” ไว้

สำหรับ “สิงหโต” คำนวณค่าความเฉพาะตัวได้ค่าเป็น 0 แต่ “สิงห” เป็นคำในพจนานุกรมจึงเก็บคำว่า “สิงห” ไว้

สำหรับ “สิงโต” คำนวณค่าความเฉพาะตัวได้ค่ามากกว่า 0 ไปขั้นตอนที่ 4 รวมกับทำเป็น “สิงโตทำ” ไปขั้นตอนที่ 5 คำนวณค่าความเฉพาะตัวได้ค่าเป็น 0 แต่ “สิงโต” เป็นคำในพจนานุกรมจึงเก็บคำว่า “สิงโต” ไว้

พิจารณา “ทำ*”

ขั้นตอนที่ 1 รวม “ทำ” กับ “ลาน” เป็น “ทำลาน”

ขั้นตอนที่ 2 คำนวณค่าความเฉพาะตัวได้ค่าเป็น 0 ไปขั้นตอนที่ 3

ขั้นตอนที่ 3 ทำการทำการจับคู่แบบประมาณ ได้ คำ “สำลาน”(เปลี่ยน “ท” เป็น“ส” ) , “ทำลาย” (เปลี่ยน “น” เป็น “ย”)

ขั้นตอนที่ 4 นำแต่ละคำรวมกับ “กรง” เป็น “สำลานกรง” และ “ทำลายกรง”

ขั้นตอนที่ 5

สำหรับ “สำลานกรง” คำนวณค่าความเฉพาะตัวได้ค่าเป็น 0 แต่ “สำลาน” เป็นคำในพจนานุกรมจึงเก็บคำว่า “สำลาน” ไว้

สำหรับ “ทำลายกรง” คำนวณค่าความเฉพาะตัวได้ค่าเป็น 0 แต่ “ทำลาย” เป็นคำในพจนานุกรมจึงเก็บคำว่า “ทำลาย” ไว้

พิจารณา “ออก*”

ขั้นตอนที่ 1 รวม “ออก” กับ “มา” เป็น “ออกมา”

ขั้นตอนที่ 2 คำนวณค่าความเฉพาะตัวได้ค่าเป็น 0 ไปขั้นตอนที่ 3

ขั้นตอนที่ 3 ทำการทำการจับคู่แบบประมาณ ได้ คำ “ออกญา”(เปลี่ยน “ม” เป็น“ญ” )

ขั้นตอนที่ 4 นำแต่ละคำรวมกับ “กรง” เป็น “กรงออกญา”

ขั้นตอนที่ 5 สำหรับ “กรงออกญา” คำนวณค่าความเฉพาะตัวได้ค่าเป็น 0 แต่ “ออกญา” เป็นคำในพจนานุกรมจึงเก็บคำว่า “ออกญา” ไว้

จากวิธีการดังกล่าว จะได้คำที่สมควรเพิ่มเข้าไปในดัชนี คือ “สิ่ง” , “สิงห” , “สิงโต” , “สำลาน” , “ทำลาย” และ “ออกญา” เมื่อเราทำการพิจารณาคำที่สงสัยว่ามีความผิดพลาด “สิ” , “ทำ” และ “ออก” ซึ่งจริงๆแล้วต้องทำกระบวนการนี้กับทุกคำที่น่าสงสัยแต่ตัวอย่างนี้แสดงเฉพาะบางส่วนซึ่งทำแล้วได้ผลลัพธ์เป็นคำออกมา การรวมกับคำข้างๆ ก็ต้องทำทั้งสองด้านเพื่อให้ได้ความแม่นยำสูง วิธีการแก้ไขยังมีจุดบกพร่องที่สามารถปรับปรุงแก้ไขให้ดีขึ้นได้โดยการพิจารณาคำที่ได้ออกมาว่าน่าจะเพิ่มคำเหล่านั้นเข้าไปในดัชนีหรือไม่ เช่น ในกรณีของการพิมพ์ผิดตัวอักษรที่นำมาแทนในตำแหน่งที่ผิดอาจจะมีตำแหน่งของปุ่มบนแป้นพิมพ์ห่างกันมาก ซึ่งการจะเพิ่มคำ

ที่เกิดจากการลักษณะนี้ไม่ค่อยสมเหตุสมผล โปรแกรมที่ผู้วิจัยได้พัฒนายังไม่มีการคำนึงถึงจุดนี้ จึงทำให้มีการเพิ่มคำที่ไม่เหมาะสมเข้าไปในดัชนีถ้าพิจารณาจากตัวอย่างจะเห็นว่าคำว่า "สิ่ง" , "สิงห" , "สำลาน" และ "ออกญา" ไม่ควรเพิ่มเข้าไปทั้งหมดถ้าเราคำนึงถึงตำแหน่งของตัวอักษรบนแป้นพิมพ์ ข้อบกพร่องนี้ทำให้ความแม่นยำของดัชนีที่ได้ค่อนข้างต่ำ (ดูค่าความแม่นยำจากผลการทดลองในบทที่ 5)

## 4.7 การทำดัชนีสำหรับข้อความภาษาไทยที่มีความผิดพลาด

ในส่วนนี้จะกล่าวถึงการทำดัชนีสำหรับข้อความภาษาไทยที่มีความผิดพลาดซึ่ง จะเน้นหนักเรื่องการดึงคำสำคัญจากข้อความ และการหาคำสำคัญที่จะเพิ่มเข้าไปในกรณีที่คาดว่าข้อความมีความผิดพลาด ซึ่งจะไม่ขอกล่าวถึงการนำคำที่ได้ไปสร้างแฟ้มดัชนีจริงๆ รหัสโปรแกรมที่ใช้ในการดึงคำสำคัญมาทำดัชนีบางส่วนสามารถดูได้ที่ ภาคผนวก ข

วิธีการทำดัชนีสำหรับข้อความภาษาไทยที่มีความผิดพลาดที่พัฒนาขึ้นนั้นจะแบ่งเป็น 2 กระบวนการคือ

1.การหาคำสำคัญจากข้อความ โดยที่ยังไม่พยายามทำการแก้ไข เนื่องจากเราไม่สามารถรู้ได้ว่าข้อความที่เข้ามามีความผิดพลาดหรือไม่ ขั้นนี้ทำเหมือนการทำดัชนีทั่ว ๆ ไป การหาคำสำคัญในส่วนนี้ อาจจะใช้วิธีการอื่นหรือวิธีการที่จะกล่าวถึงต่อไปก็ได้

2.การหาคำสำคัญที่จะเพิ่มเข้าไปในดัชนีสำหรับกรณีที่คาดว่าข้อความมีความผิดพลาด ขั้นตอนนี้จะทำการวิเคราะห์ข้อความว่ามีความเป็นไปได้หรือไม่ที่ในข้อความจะมีความผิดพลาดแฝงอยู่ ถ้าพบว่าเป็นไปได้ว่าจะมีความผิดพลาดอยู่ก็จะทำการหาคำสำคัญเพิ่มเติมเพื่อที่จะเพิ่มเข้าไปในดัชนีเพื่อให้ได้ดัชนีที่มีความสมบูรณ์ขึ้น

### 4.7.1 การหาคำสำคัญจากข้อความ

กระบวนการที่คิดค้นขึ้นจะเริ่มจาก การแบ่งคำซึ่งใช้วิธีการดังที่กล่าวในหัวข้อที่ 4.4 ซึ่งจุดหนึ่งที่ต้องคำนึงถึงก็คือ การหาค่าความเฉพาะตัวของคำซึ่งจะถูกใช้บ่อยและจำเป็นต้องทำให้เร็ว ซึ่งผู้วิจัยได้ประยุกต์โครงสร้างข้อมูลแบบทรัย ให้สามารถทำการหาค่าความเฉพาะตัวของคำได้อย่างรวดเร็ว โดยใช้เวลาเป็น O(ความยาวของคำ) รายระเอียดเรื่องการประยุกต์โครงสร้างข้อมูลแบบทรัย เพื่อหาค่าความเฉพาะตัวของคำที่กล่าวในหัวข้อที่ 4.4 เมื่อได้ข้อความที่แบ่งเป็นคำแล้วก็จะวิเคราะห์ความเป็นไปได้ที่คำ 2 คำที่อยู่ติดกันมีความกำกวม ยกตัวอย่างเช่น "ตากลม" ซึ่งเมื่อผ่านกระบวนการแบ่งคำที่กล่าวในหัวข้อที่ 4.4 จะได้ "ตาก" และ "ลม" แต่เนื่องจาก "ตาก" และ "ลม" มีความกำกวมคือ สามารถใช้จุดอื่นในการแบ่งคำและผลที่ได้จากการแบ่งเป็นคำในพจนานุกรมให้ทั้งหมดในกรณีนี้คือ "ตา" และ "กลม" ให้นำคำที่ได้นี้รวมกับคำที่ได้จากการแบ่งคำที่เป็นคำในพจนานุกรม เป็นคำที่ควรนำไปทำดัชนี

ตัวอย่างที่ 4 "ชาวนากับงูเห่า" ข้อสังเกตุไม่มีคำว่า "ชาวนา" ในพจนานุกรม
แบ่งคำได้เป็น "ชา" , "วนา" , "กับ" , "งูเห่า"
คำที่ได้จากความกำกวม "ชาว" , "นา"
คำที่ได้จากการแบ่งคำที่อยู่ในพจนานุกรม "ชา" , "วนา" , "กับ" , "งูเห่า"
คำที่ควรนำไปทำดัชนี "ชา" , "วนา" , "กับ" , "งูเห่า" , "ชาว" , "นา"

ตัวอย่างที่ 5 "ผู้ประกอบยการ"
แบ่งคำได้เป็น "ผู้" , "ประกอบ" , "ย" , "การ"

คำที่ได้จากความกำกวม ไม่มี

คำที่ได้จากการแบ่งคำที่อยู่ในพจนานุกรม "ผู้" , "ประกอบ" , "การ"
คำที่ควรนำไปทำดัชนี "ผู้" , "ประกอบ" , "การ"

4.7.2 การหาคำสำคัญที่จะเพิ่มเข้าไปในดัชนีสำหรับกรณีที่คาดว่าข้อความมีความผิดพลาด

ส่วนนี้ทำโดยเริ่มจากการแบ่งคำไทยซึ่งเป็นวิธีการเดียวกับที่กล่าวในหัวข้อที่ 4.4 แล้วใช้การค้นหาความผิดพลาดในข้อความภาษาไทยซึ่งเป็นวิธีการเดียวกับที่กล่าวในหัวข้อที่ 4.5 เมื่อพบจุดที่คาดว่าเป็นความผิดพลาดก็จะใช้การแก้ไขความผิดพลาดในข้อความภาษาไทยไทยซึ่งใช้วิธีการเดียวกับที่กล่าวในหัวข้อที่ 4.6 เพื่อสร้างคำสำคัญที่จะเพิ่มเข้าไปในดัชนีสำหรับกรณีที่คาดว่าข้อความมีความผิดพลาด

ตัวอย่างที่ 6 "นายกรัฐนมตรีเดินทางไปต่างประเทศ"
แบ่งคำได้เป็น "นายก" , "รัฐ" , "นม*" , "ตรี*" , "เดินทาง" , "ไป*" , "ต่าง" , "ประเทศ"
"*" หมายถึงคำที่อาจเกิดจากความผิดพลาด
คำที่ควรเพิ่มเข้าไปในดัชนี "นายกรัฐมนตรี" , "ไมตรี"

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

# บทที่ 5

## ผลการทดลอง

ในบทนี้จะกล่าวถึงผลการทดลองของขั้นตอนวิธีการจัดทำดัชนีสำหรับข้อความไทยที่มีความผิดพลาดที่ได้คิดค้นขึ้น การทดสอบเน้นเรื่องความสามารถในการดึงคำออกมาจากข้อความเป็นหลักโดยการวัดค่าจะทำโดยใช้วิธีการที่จะกล่าวในหัวข้อที่ 5.1 โดยการทดสอบจะทำกับข้อความซึ่งนำมาจาก ข่าว และ วิทยานิพนธ์ โดยจำนวนข้อความที่นำมาทดสอบมีทั้งหมดมีขนาด 1800 ประโยค ตัวอย่างของข้อความที่นำมาทดสอบสามารถดูได้จากภาคผนวก ค.

### 5.1 การวัดผลการทดลอง

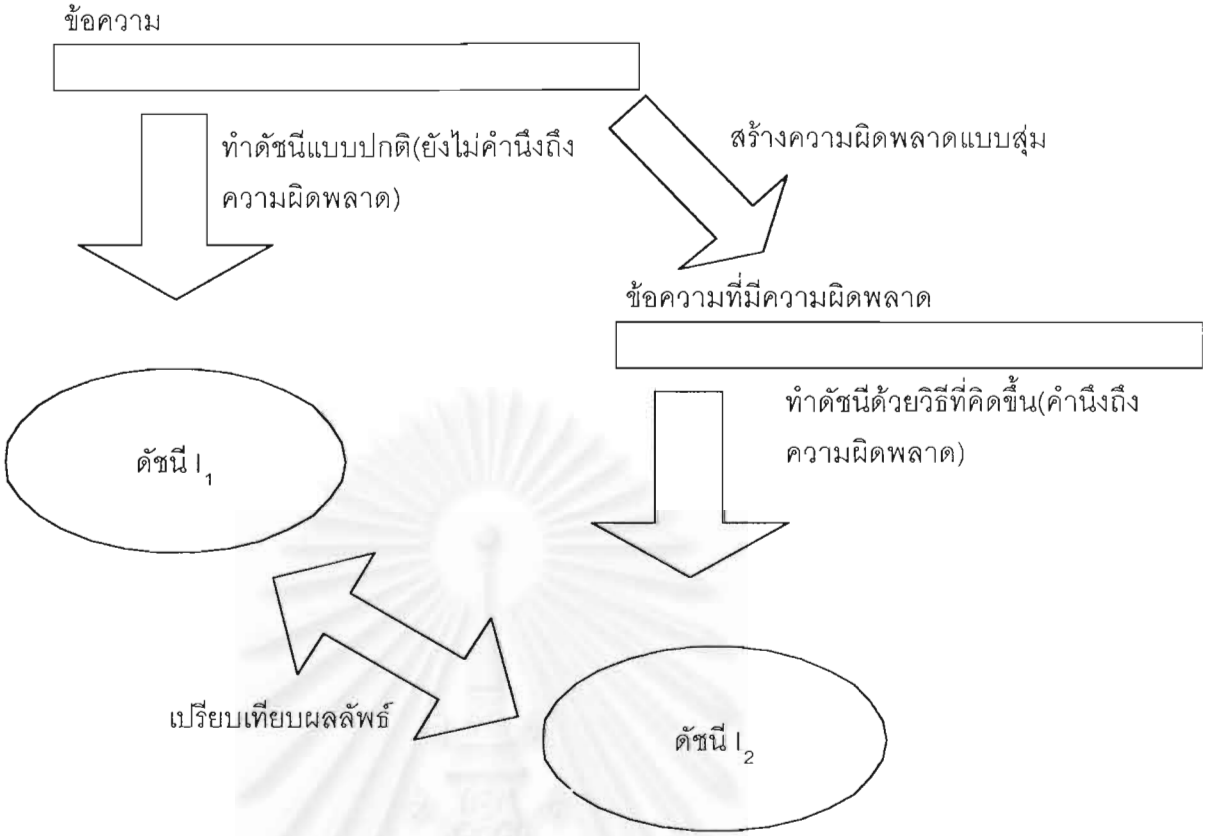
วิธีการการวัดผลการทำงานของโปรแกรมจะทำโดยนำข้อความปกติมาทำดัชนีและเก็บคำที่ได้เอาไว้เพื่อใช้ในการเปรียบเทียบหลังจากนั้นนำข้อความนี้ไปสร้างความผิดพลาดแบบสุ่มแล้วใช้วิธีการที่คิดขึ้นทำดัชนีแล้วทำการเปรียบเทียบคำสำคัญที่ได้จากทั้ง 2 วิธี ดังแสดงในรูปที่ 5.1 โดยจะวัดผลเป็นค่า 2 ค่าดังนี้

1.เปอร์เซ็นต์ของจำนวนคำที่ได้จากวิธีการทำดัชนีที่คิดขึ้นที่มีในดัชนีที่ได้จากข้อความปกติและการทำดัชนีแบบปกติ ( $|I_1 \cap I_2|$ ) ต่อ จำนวนคำทั้งหมดที่ได้จากข้อความปกติและการทำดัชนีแบบปกติ ( $|I_1|$ ) จะขอเรียกค่านี้ว่า ความสมบูรณ์ของดัชนี (completeness) ซึ่งค่านี้จะขอแยกออกเป็น 2 ค่าคือ ค่าความสมบูรณ์เมื่อนำข้อความที่มีความผิดพลาดมาทำดัชนีโดยยังไม่เพิ่มคำที่สมควรเพิ่มเข้าไป และ ค่าความสมบูรณ์หลังจากได้เพิ่มคำที่สมควรเพิ่มไปแล้ว

$$\text{ความสมบูรณ์} = \frac{|I_1 \cap I_2|}{|I_1|} \times 100$$

2.เปอร์เซ็นต์ของจำนวนคำที่ได้จากวิธีการทำดัชนีที่คิดขึ้นที่มีในดัชนีที่ได้จากข้อความปกติและการทำดัชนีแบบปกติ( $|I_1 \cap I_2|$ ) ต่อจำนวนคำทั้งหมดที่ได้จากวิธีการที่คิดขึ้น( $|I_2|$ ) จะขอเรียกค่านี้ว่า ความแม่นยำของดัชนี ( precision ) ซึ่งค่านี้จะขอแยกออกเป็น 2 ค่าคือ ค่าความแม่นยำเมื่อนำข้อความที่มีความผิดพลาดมาทำดัชนีโดยยังไม่เพิ่มคำที่สมควรเพิ่มเข้าไป และ ค่าความสมบูรณ์หลังจากได้เพิ่มคำที่สมควรเพิ่มไปแล้ว

$$\text{ความแม่นยำ} = \frac{|I_1 \cap I_2|}{|I_2|} \times 100$$



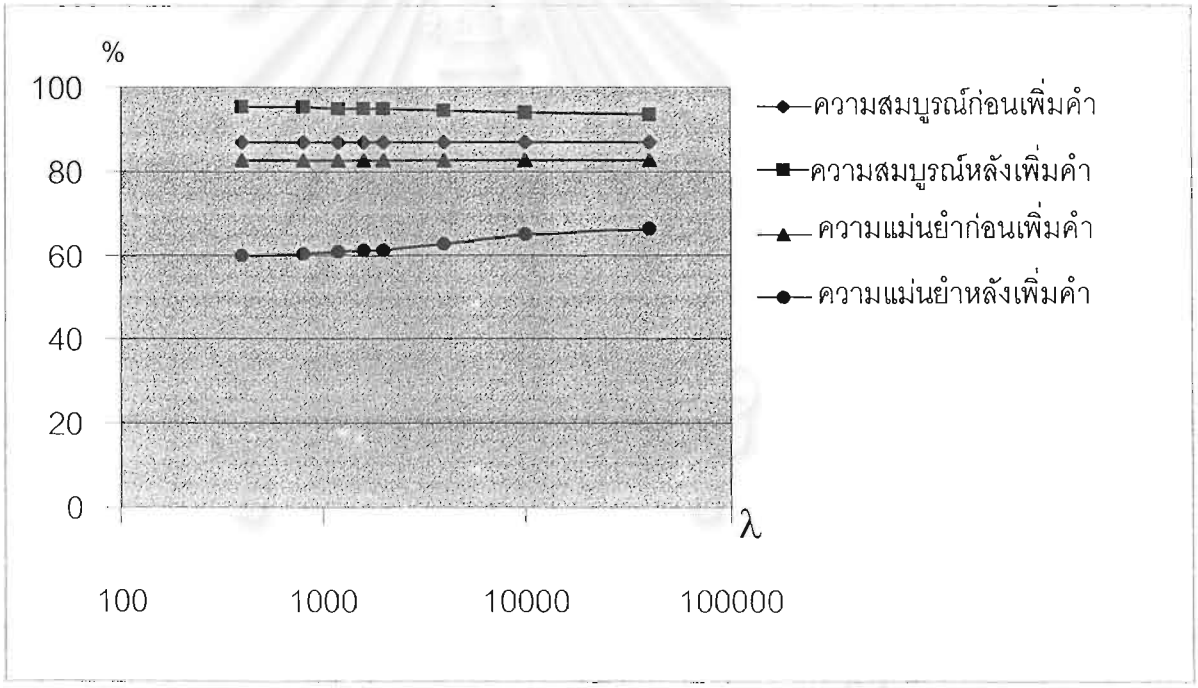

**รูปที่ 5.1 แสดงวิธีการวัดผล**

การทดสอบจะทำโดยเปลี่ยนแปลงค่า $\lambda$ ซึ่งเป็นตัวแปรที่ใช้ในการค้นหาความผิดพลาด ค่าที่ต่ำจะค้นหาความผิดพลาดได้มากแต่ก็ช้าและอาจคิดว่าส่วนของข้อความที่ไม่มีความผิดพลาดบางส่วนเป็นความผิดพลาด ในขณะที่ค่าที่สูงจะทำงานได้เร็วแต่ก็จะมองข้ามความผิดพลาดบางส่วนไป ค่าพารามิเตอร์อื่นที่ใช้จะมีค่าดังนี้คือ $\beta = 20$ และ $\gamma = 10$ การเลือกคำมาทำดัชนีนั้นจะเลือกเฉพาะคำที่มีความยาวมากกว่า 2 ตัวอักษรสำหรับคำที่ออกมาจากข้อความโดยไม่ผ่านการแก้ไข และ เลือกเฉพาะคำที่มีความยาวมากกว่า 3 ตัวอักษรสำหรับคำที่ผ่านการแก้ไข ผลลัพธ์ที่ได้แสดงดังตารางที่ 5.1 และรูปที่ 5.2 ข้อสังเกตค่าความแม่นยำหลังจากเพิ่มคำเข้าไปแล้วจะต่ำเนื่องจากยังไม่มีการคัดเลือกคำที่ได้จากการแก้ไปที่ดีพอซึ่งเป็นจุดหนึ่งที่สามารถพัฒนาต่อได้

การวัดผลที่น่าสนใจอีกอย่างหนึ่งคือค่าเปอร์เซ็นที่วิธีที่คิดขึ้นสามารถหาคำทั้งหมดที่ได้จากข้อความก่อนที่จะทำให้เกิดความผิดพลาดด้วยวิธีการทำดัชนีปกติ โดยการวัดค่านี้จะนับเป็นเปอร์เซ็นต์ของประโยคที่ใช้วิธีการทำดัชนีที่คิดขึ้นแล้วได้ดัชนีที่มีคำทั้งหมดที่มีในดัชนีจากข้อความก่อนที่จะทำให้เกิดความผิดพลาด ( $I_1 \cap I_2 = I_1$ ) โดยใช้พารามิเตอร์ต่างๆในการวัดผลเหมือนกับการทดสอบข้างต้น ผลลัพธ์ที่ได้แสดงดังตารางที่ 5.2 และรูปที่ 5.3

**ตารางที่ 5.1 แสดงผลการทดลองการการดึงคำเพื่อไปทำดัชนีด้วยวิธีที่คิดค้นขึ้น**

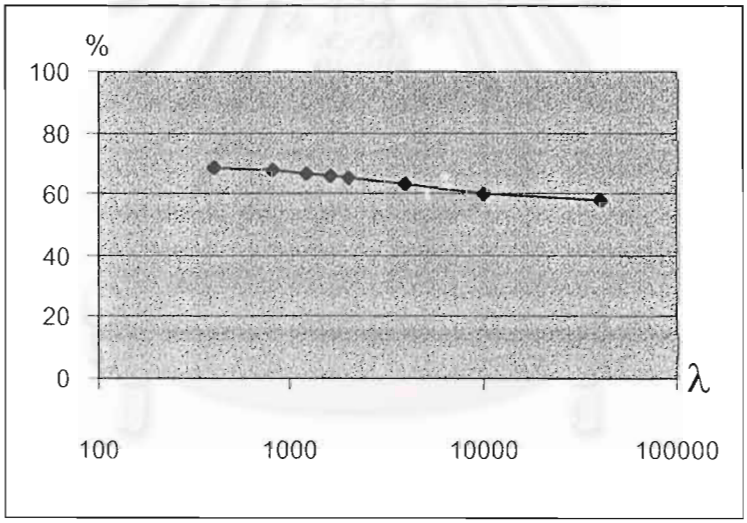
| λ | ความสมบูรณ์(%) | | ความแม่นยำ(%) | |
|---|---|---|---|---|
| | ก่อนเพิ่มคำ | หลังเพิ่มคำ | ก่อนเพิ่มคำ | หลังเพิ่มคำ |
| 400 | 87.12 | 95.36 | 82.91 | 59.92 |
| 800 | 87.12 | 95.20 | 82.91 | 60.43 |
| 1200 | 87.12 | 95.01 | 82.91 | 60.62 |
| 1600 | 87.12 | 94.93 | 82.91 | 60.99 |
| 2000 | 87.12 | 94.82 | 82.91 | 61.29 |
| 4000 | 87.12 | 94.40 | 82.91 | 62.54 |
| 10000 | 87.12 | 93.99 | 82.91 | 64.95 |
| 40000 | 87.12 | 93.55 | 82.91 | 66.18 |




**รูปที่ 5.2 แสดงผลการทดลองการการดึงคำเพื่อไปทำดัชนีด้วยวิธีที่คิดค้นขึ้น**

**ตารางที่ 5.2 แสดงเปอร์เซ็นต์ของประโยคที่ใช้วิธีการทำดัชนีที่คิดขึ้นแล้วได้ดัชนีที่มีคำทั้งหมดที่มีในดัชนีจากข้อความก่อนที่จะทำให้เกิดความผิดพลาด**

| λ | เปอร์เซ็นต์ของประโยคที่วิธีการที่คิดขึ้นสามารถหาคำทั้งหมดที่ได้จากดัชนีของข้อความก่อนที่จะทำให้เกิดความผิดพลาด |
|---|---|
| 400 | 68.85 |
| 800 | 67.73 |
| 1200 | 66.47 |
| 1600 | 66.11 |
| 2000 | 65.57 |
| 4000 | 63.15 |
| 10000 | 60.32 |
| 40000 | 57.99 |




**รูปที่ 5.3 แสดงเปอร์เซ็นต์ของประโยคที่ใช้วิธีการทำดัชนีที่คิดขึ้นแล้วได้ดัชนีที่มีคำทั้งหมดที่มีในดัชนีจากข้อความก่อนที่จะทำให้เกิดความผิดพลาด**

## 5.2 สรุปผลการทดลอง

การผลการทดลองข้างต้นวิธีการที่คิดค้นขึ้นสามารถช่วยดึงคำออกจากข้อความที่มีความผิดพลาดได้ และจะสามารถช่วยให้สามารถสร้างดัชนีที่มีความสมบูรณ์มากขึ้นได้ถึงแม้จะไม่มากก็ตามกล่าวคือประมาณ 7 เปอรเซ็นต์ เหตุผลที่ค่าความสมบูรณ์ของดัชนีเพิ่มขึ้นเพียง 7 เปอร์เซ็นต์นั้นเนื่องจากการวัดผลนั้นทำโดยการสร้างความผิดพลาดขึ้น 1 ตำแหน่งในประโยคดังนั้นจะมีส่วนที่ถูกต้องเยอะอยู่แล้วพิจารณาจากตารางจะเห็นว่าความสมบูรณ์ของดัชนีก่อนเพิ่ม

คำนั้นเป็น 87 เปอร์เซ็นต์อยู่แล้ว ซึ่งหมายถึงความผิดพลาดทำให้ความสมบูรณ์ของดัชนีลดลงประมาณ 13 เปอร์เซ็นต์ แต่หลังจากทำการเพิ่มคำจำได้ความสมบูรณ์ส่วนที่หายไปกลับคืนมาประมาณ 7 เปอร์เซ็นต์ นั่นคือวิธีการนี้ช่วยจัดการกับความผิดพลาดได้เท่ากับ 7/13×100% หรือประมาณ 53 เปอร์เซ็นต์ เหตุผลที่ค่านี้ยังไม่สูงมากเนื่องจากขั้นตอนวิธีที่ใช้นั้นยังพิจารณาแก้ไขเฉพาะกรณีที่เมื่อพบความผิดพลาดและทำการจับคู่แบบประมาณแล้วจะทำการรวมกับคำข้างเคียงทั้งคำว่าเกิดเป็นคำใหม่ได้หรือไม่ โดยไม่พิจารณาการรวมกับเพียงบางส่วนของคำข้างเคียง เช่น “หอมอบอวล” แต่พิมพ์ผิดเป็น “หอมอบดอวล” เมื่อทำการแบ่งคำจะได้ “หอ*” , “มอ*” , “บด*” , “อวล” ซึ่งพบว่าถ้าพิจารณาความผิดพลาดของ “บด*” และทำการจับคู่แบบประมาณของ “บด*” กับ “อวล” จะได้ “บอวล” ซึ่งเมื่อนำมารวมกับ “มอ*” แล้วได้เป็น “มอบอวล” ซึ่งไม่เป็นคำจึงทิ้ง “บอวล” ไปเนื่องจาก “บอวล” ก็ไม่เป็นคำที่สมบูรณ์ ในขณะที่ถ้ารวม “บอวล” กับ “อ” ซึ่งเป็นส่วนหนึ่งของ “มอ*” ก็จะได้คำว่า “อบอวล” ออกมา อย่างไรก็ดีการรวมแบบดังกล่าวจะเพิ่มการคำนวณขึ้นมากเพราะต้องทำการรวมเป็นจำนวนเท่ากับความยาวเป็นตัวอักษรของคำข้างเคียงผู้วิจัยจึงได้ไม่ใส่การประมวลผลในลักษณะนี้เข้าไปในขั้นตอนวิธีที่คิดขึ้น และ อีกกรณีหนึ่งก็คือความผิดพลาดไม่แสดงออกมาให้วิธีการค้นหาความผิดพลาดที่คิดค้นขึ้นสังเกตุเห็นได้เลยเช่น “กรมการปกครอง” พิมพ์ผิดเป็น “กรมการกครอง” ซึ่งเมื่อผ่านการแบ่งคำจะได้ “กรม” , “การก” , “ครอง” ซึ่งไม่แสดงความผิดพลาดออกมาเลยดังนั้นจึงไม่สามารถหาคำเพิ่มเติมได้ในกรณีเช่นนี้

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

# บทที่ 6

# บทสรุปและข้อเสนอแนะ

จุดประสงค์ในการทำดัชนีสำหรับข้อความไทยที่มีความผิดพลาดนั้นก็คือความสามารถที่จะเพิ่มคำที่ถูกต้องเข้าไปในดัชนีได้เมื่อข้อความมีความผิดพลาดปนอยู่ แต่เนื่องจากคำภาษาไทยมีรูปแบบการเขียนที่ติดกันทำให้มีปัญหาเรื่องขอบเขตของคำส่งผลให้การใช้วิธีการค้นหาความผิดพลาดและการการแก้ไขความผิดพลาดเหมือนกับที่ใช้ในภาษาอังกฤษไม่เหมาะสม ดังนั้นจึงจำเป็นต้องพัฒนาวิธีหาความผิดพลาดและแก้ไขความผิดพลาดที่เหมาะกับลักษณะของภาษาไทยเพื่อที่จะสามารถค้นหาความผิดพลาดและสร้างคำที่เหมาะสมเพื่อเพิ่มเข้าไปในดัชนีเพื่อให้ได้ดัชนีที่มีความสมบูรณ์มากขึ้น

## 6.1 สรุป

กระบวนการทำดัชนีสำหรับข้อความไทยที่มีความผิดพลาดสามารถแบ่งเป็นขั้น ๆ ได้ดังนี้

ขั้นตอนที่ 1 การแบ่งคำ ขั้นตอนนี้จะนำข้อความมาตัดออกเป็นคำเพื่อความเหมาะสมในการค้นหาความผิดพลาด

ขั้นตอนที่ 2 การหาคำสำคัญจากข้อความ ขั้นตอนนี้จะทำเหมือนการทำดัชนีปกติคือหาคำที่สมควรจะนำไปทำดัชนี โดยขั้นตอนนี้ยังไม่คำนึงถึงความผิดพลาด

ขั้นตอนที่ 3 การหาคำสำคัญที่จะเพิ่มเข้าไปในดัชนีสำหรับกรณีที่คาดว่าข้อความมีความผิดพลาด ขั้นตอนนี้จะทำการหาคำสำคัญที่จะเพิ่มเข้าไปในดัชนีเพื่อให้ได้ดัชนีที่มีความสมบูรณ์มากขึ้น ในกรณีที่คาดว่าข้อความมีความผิดพลาด ขั้นตอนนี้สามารถแบ่งเป็นขั้นตอนย่อยๆ ได้ดังนี้

ขั้นตอนที่ 3.1 การค้นหาความผิดพลาด ขั้นนี้จะเป็นขั้นตอนที่จะค้นหาความผิดพลาดเพื่อใช้ในการแก้ไขความผิดพลาดในขั้นตอนต่อไป โดยขั้นตอนนี้จะใช้ค่าความเฉพาะตัวของแต่ละสตริงเป็นหลักในการบ่งชี้ว่าสตริงนั้น น่าจะเป็นความผิดพลาดหรือไม่

ขั้นตอนที่ 3.2 การแก้ไขความผิดพลาด ขั้นตอนนี้จะทำการหาคำที่สมควรจะเพิ่มเข้าไปในดัชนีในกรณีที่คาดว่าในข้อความมีความผิดพลาด

หลังจากนั้นก็นำคำที่ได้จาก ขั้นตอนที่ 2 และ ขั้นตอนที่ 3 ไปทำดัชนี โดยอาจนำไปผ่านกระบวนการคัดเลือกคำก่อนที่จะนำไปเก็บในแฟ้มดัชนีจริงๆ หรือไม่นั้นก็ขึ้นอยู่กับความเหมาะสมกับงานที่จะนำไปใช้

## 6.2 ข้อเสนอแนะ

วิทยานิพนธ์นี้ได้ทำให้เกิดแนวความคิดในการวิจัยอื่นๆ ต่อไปอีก เช่น

- นำแนวคิดนี้ไปประยุกต์ใช้กับโปรแกรมประมวลผลคำ เพื่อใช้ในการค้นหาความผิดพลาดและการแนะนำคำที่เหมาะสมที่จะใช้แก้ความผิดพลาด
- การพัฒนาขั้นตอนวิธีเพิ่มเติมเพื่อให้สามารถทำงานได้กับความผิดพลาดที่มากขึ้น เพราะขั้นตอนวิธีที่พัฒนาขึ้นนี้รองรับเพียงแค่ ความผิดพลาดแบบตำแหน่งเดียวในคำ
- สร้างพจนานุกรมภาษาไทย ที่มีจำนวนคำที่เหมาะสมสำหรับขั้นตอนวิธีที่พัฒนาขึ้น

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

# รายการอ้างอิง


1. William B. Frakes and Ricardo Baeza-Yates , Information Retrieval Data Structures & Algorithms , (USA;Pentice Hall Inc;1992)

2. Karen Kukick , "Techniques for Automatically Correcting Words in Text", ACM Computing Surveys,Vol.24 No.4 pp 377-439

3. Witoon Kanlayanawat and Somchai Prasitjutrakul , "Automatic Indexing for Thai Text with Unknown Words using Trie Structure" , *Natural Language Processing Pacific Rim Symposium 1997 (NLPRS'97), Phuket, Thailand, December 2-4,1997.*

4. วิฑูรย์ กัลยาณวัฒน์ , "ระบบการค้นคืนข้อความภาษาไทยโดยใช้แฟ้มข้อมูลผกผัน" วิทยานิพนธ์ปริญญามหาบัณฑิต ภาควิชาวิศวกรรมศาสตร์ บัณฑิตวิทยาลัย จุฬาลงกรณ์มหาวิทยาลัย , 2540

5. Ralph P.Grimaldi . Discrete and Combinatorial Mathematics . USA : Addison-Wesley Publishing Company Inc , 1994

6. H. Shang and T.H. Merrett, "Tries for Approximate String Matching," *IEEE Trans. on Knowledge and Data Eng.*, Vol. 8 , No. 4 , Aug. 1996 , pp. 540-547.


จุฬาลงกรณ์มหาวิทยาลัย

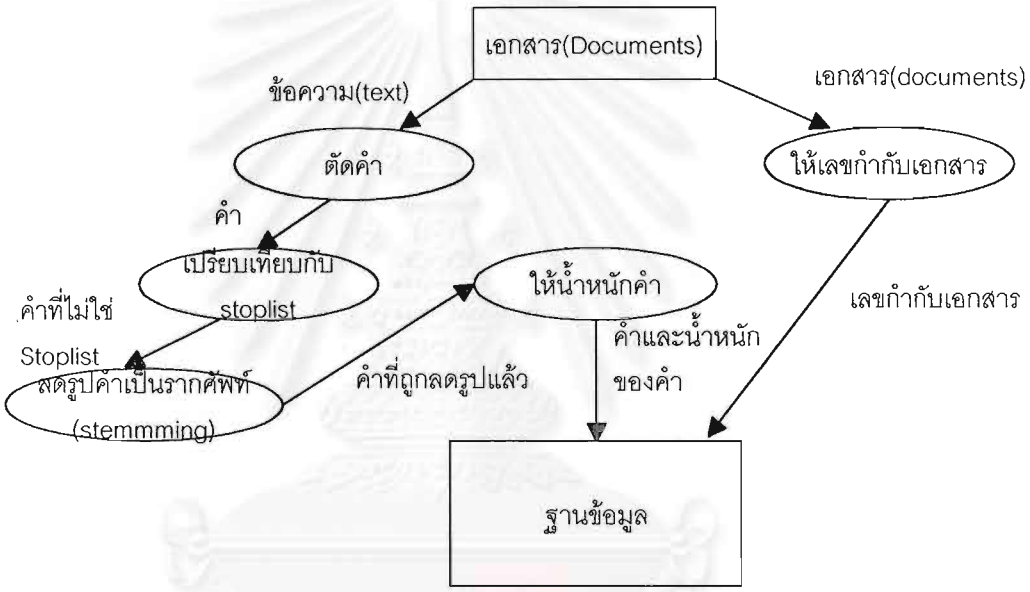
# ภาคผนวก

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

# ภาคผนวก ก.

**การจัดทำดัชนี** (Indexing)

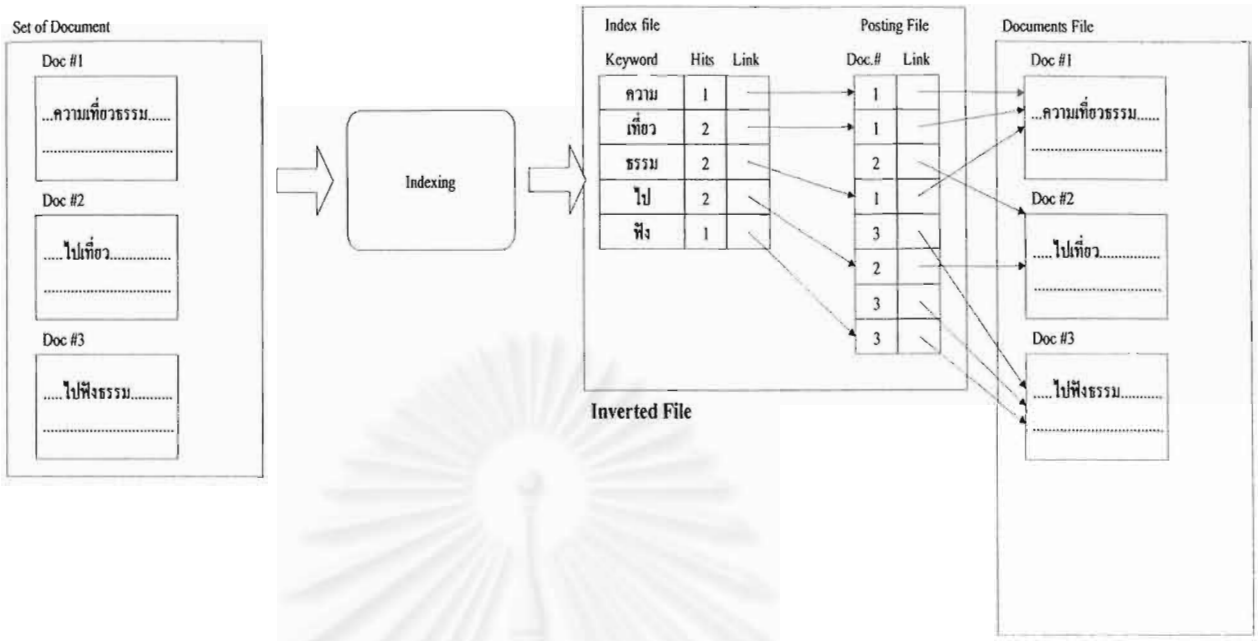
การจัดทำดัชนีจะมีขั้นตอนดังแสดงในรูปที่ ก.1 เริ่มจากให้หมายเลขอ้างอิงสำหรับเอกสาร นำเอกสารมาตัดเป็นคำ นำคำที่ได้เหล่านั้นมาตัดคำที่ไม่สำคัญออกไปโดยมาเทียบกับคำที่ไม่น่าจะนำไปทำดัชนีสำคัญ (Stop list) [1] หลังจากนั้นนำคำที่เหลือมลดรูปคำเป็นรากศัพท์ (Stemming)[1]เพื่อให้เป็นรูปแบบที่มาตราฐานและทำการค้นหาได้ดีขึ้น ขั้นตอนสุดท้ายคือการให้ความสำคัญกับคำต่างๆเพื่อที่จะจัดลำดับความสำคัญให้กับเอกสารที่ค้นคืนมาได้ในภายหลัง และการให้หมายเลขกำกับเอกสารเพื่อใช้ในการอ้างอิง รูปที่ ก.2 แสดงการนำเอกสารมาจัดทำดัชนีและเก็บโดยใช้แฟ้มข้อมูลผกผัน




**รูปที่ ก.1 แสดงขั้นตอนการจัดทำดัชนี**[1]

**โครงสร้างแฟ้มข้อมูลผกผัน**

โครงสร้างแฟ้มข้อมูลผกผันแบ่งเป็น 2 ส่วนคือ Index File และ Posting File ดังแสดงในรูปที่ ข.2 การสร้างแฟ้มข้อมูลผกผันนั้นเริ่มจากการดึงคำสำคัญพร้อมตำแหน่งของคำสำคัญนั้นในเอกสารออกมาจากเอกสารทุกฉบับในระบบนำมาเรียงลำดับ นำคำสำคัญและจำนวนเอกสารที่คำสำคัญนั้นปรากฏ(Hits) รวมทั้งตำแหน่งใน Posting File (Link) เพื่อใช้ในการอ้างถึงไว้ใน Index File และเก็บหมายเลขเอกสารและตำแหน่งที่คำสำคัญนั้นปรากฏในเอกสารไว้ใน Posting File




**รูปที่ ก.2 แสดงคำสำคัญที่ได้จากการจัดทำดัชนีและเก็บเป็นแฟ้มข้อมูลผกผัน**

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

# ภาคผนวก ข

## รหัสโปรแกรม ส่วนที่สำคัญ

```
/*------ data structure use to hold parameter of program -------*/
typedef struct config_var
              {
                // dictionary parameter
                int total_word_in_dict;
                // n-gram  and trie parameter
                int ngram_dimension;
                int start_char_ascii_no;
                int last_char_ascii_no;
              // tuning factor here
                // weight relate factor use to calculate weight for each string
                int alpha_factor;
                int beta_factor;
                int gamma_factor;
                int delta_factor;
                int theta_factor;
                int zigma_factor;
                // factor need when processing error
                // correction use to calculate threshold
                int require_split_length;
                int AA;
                int BB;
                int CC;
                // minimum length of sting require to perform approximate match
                int  min_insertion_error;
                int  min_deletetion_error;
                int  min_substitutetion_error;
                int  min_transposition_error;
                // indexing parameter
                // may considered as filter use to decide accept word as index or not
                int  min_original_accept_length;
                int  min_fixed_accept_length;

              }CONFIG_VAR;

/*------ data structure use to keep information of each word in dictionary -------*/
typedef struct my_word
            {
              unsigned char w[MAXWORDLENGTH];
              int wlength;
              unsigned char flag[NUMOFFLAG];
              // description of each flag here
              // flag 0 word have space inside
              // flag 1 prefix word probably connect to the following word
              // flag 2 suffix word probably connect to former word
              // flag 3 word have maiyamok inside
              // flag 4 single letter word
              // flag 5 special case have only 1 word of this type in dictionary
              // flag 6 word have many meanings
              // flag 7 able to be stand alone word
              //         most word is stand alone except some word that
              //         make flag prefix or suffix turned on
              // flag 8 packed word,word deleted space inside
              // flag 9 fraction word, seperate fraction part of word
              //        that have space inside
              // flag 10 replaced maiyamok, replace occurance of maiyamok
              // flag 11-15 reserve (make record size 80 for easy view with
              //                     hex editor)

            }MY_WORD;

/*-------- set ADT -----------*/
// Set have dummy node
// Field word_number of dummy node also hold set size
typedef struct node {
                      WORD_NUMBER word_number;
                      struct node *next;
                    }NODE;
```

```
typedef struct set {
                      NODE *head;
                      NODE *tail;
                    }SET;


void create_node(WORD_NUMBER word_number,NODE **n)
{
  NODE *temp;
  temp = (NODE *)(malloc(sizeof(NODE)));
  temp->word_number = word_number;
  temp->next = NULL;
  *n = temp;
  number_of_nodes_used++;
}

void init_set(SET *s)
{// create empty set with dummy node
  NODE *temp;
  create_node(0,&temp);
  s->head = temp;
  s->tail = temp;
}

void free_set(SET *s)
{
  NODE *current,*tmp;
  current = s->head;
  while(current->next != NULL)
  {
    tmp = current->next;
    current->next = current->next->next;
    free(tmp);
    number_of_nodes_used--;
  }
  free(s->head); // free dummy node
  number_of_nodes_used--;
}

int empty_set(SET *s)
{
  if (s->head == s->tail)
    return 1;
  else
    return 0;
}

void delete_set(SET *s,NODE *n)
{
  NODE *tmp;
  tmp = n->next;
  n->next = tmp->next;
  // not need to update head coz it always point to dummy node
  if (s->tail == tmp)// require to update tail if node delete is last
  {
    s->tail = n;
  }
  s->head->word_number--;//decrease list size by 1
  free(tmp);
  number_of_nodes_used--;
}


void insert_set(SET *s,NODE *n)
{
  // assumming  element insert to list is comming in order

  if (empty_set(s))
  {
    s->head->next = n;
    s->tail = n;
    s->head->word_number++;//increase list size by 1
  }
  else
  {
    if(n->word_number > s->tail->word_number)
    {
      s->tail->next = n;
      s->tail = n;
```

```
      s->head->word_number++;//increase list size by 1
    }
    else
    {
      free(n);
      number_of_nodes_used--;
    }
  }
}


/*-------- group ADT -----------*/
//hold information of vertice in graph
typedef struct group
            {
              char str[MAXWORDLENGTH];
              int start_char;
              int stop_char;
              SET *s;
              int group_weight;
            }GROUP;


void init_group(GROUP *g)
{
  sprintf(g->str,"");
  g->start_char = -1;
  g->stop_char = -1;
  g->group_weight = 0;
  g->s = (SET *)(NULL);
}

void setup_group(GROUP *g,char *str,int start_char,int stop_char)
{
  strcpy(g->str,str);
  g->start_char = start_char;
  g->stop_char  = stop_char;
}

void free_group(GROUP *g)
{
  free_set(g->s);
  free(g->s);
}

void make_group(CONFIG_VAR *cfg,GROUP *g,SET *s,
                char *str,int start_char,int stop_char,
                MY_WORD *dictionary)
{

  int len;
  int a_factor=1;
  int alpha_factor = cfg->alpha_factor;
  int beta_factor  = cfg->beta_factor;
  int gamma_factor = cfg->gamma_factor;
  int delta_factor = cfg->delta_factor;
  int theta_factor = cfg->theta_factor;
  int zigma_factor = cfg->zigma_factor;
  int num_of_associate_word;

  strcpy(g->str,str);
  len = strlen(g->str);
  g->start_char = start_char;
  g->stop_char  = stop_char;
  g->s = s;

  if (len == 1)
  {
    num_of_associate_word = 0;
  }
  else
  {
    num_of_associate_word = g->s->head->word_number;
  }




if ( is_in_dictionary(dictionary,cfg->total_word_in_dict,str) >= 0 )
```

```
    a_factor = 0;
   // init group weight here
  if (num_of_associate_word == 0)
  {
    if (len == 1)
    {
      g->group_weight = delta_factor+(int)(1.0*beta_factor*1);
    }
    else
    {
      g->group_weight = zigma_factor;
    }
  }
  else
  {
    g->group_weight =  (a_factor*delta_factor)+
                       (int)((1.0*beta_factor*num_of_associate_word)/len);
  }
}

int group_type(CONFIG_VAR *cfg,GROUP *g)
{
  int ret = 0;
  int num_of_associate_word = g->s->head->word_number;

  if (num_of_associate_word == 0)
  {
    if ( strlen(g->str) == 1 )
    {
      ret = 2;
    }
    else
    {
      ret = 1;
    }
  }
  else
  {
    if ( g->group_weight >= (cfg->alpha_factor*cfg->delta_factor) )
    {
      ret = 3;
    }
    else
    {
      ret = 4;
    }
  }


  return (ret);
}


void group_copy(GROUP *destination,GROUP *source)
{
  memcpy(destination,source,sizeof(GROUP));
}


/*-------- graph ADT ----------*/
typedef struct graph
          {
            GROUP *vertice[MAXGROUPINLINE];
            int   edge[MAXGROUPINLINE][MAXGROUPINLINE];
            int   num_of_vertice;
          }GRAPH;

void init_graph(GRAPH *gph,int num_of_group)
{
  int i,j;
  int num_of_vertice = num_of_group+2;

  for(i=0;i<num_of_vertice;i++)
  {
    gph->vertice[i] = (GROUP *)(NULL);
    for(j=0;j<num_of_vertice;j++)
    {
      gph->edge[i][j] = -1; // use -1 as empty edge
    }
```

```
  }
  gph->num_of_vertice = 0;
}


void crate_graph(GRAPH *gph,GROUP *g[],int num_of_group)
{
  // vertice #0 and vertice #(num_of_syl+1) are dummy vertice
  // graph is directed graph then edge[i][j]  is from i(source vertice) to j
(destination vertice)
  int i,j,last_char = -1;

  gph->num_of_vertice = num_of_group+2;

  //finding last_char : need to use to short vertice to last dummy vetice
  for(i=0;i<num_of_group;i++)
  {
    if (g[i]->stop_char > last_char)
      last_char = g[i]->stop_char;
  }

  // create vertice set
  // vertice set have 2 dummy node one at the beginning and one at ending
  for(i=0;i<num_of_group;i++)
  {
    // create vertice
    gph->vertice[i+1] = g[i];
  }

  for(i=0;i<num_of_group;i++)
  {
    if (g[i]->start_char == 0)
    { // create link from first dummy node to vertice i
      gph->edge[0][i+1] = g[i]->group_weight;
    }

    if (g[i]->stop_char == last_char)
    { // create link from vertice i to last dummy node
      gph->edge[i+1][num_of_group+1] = 0;
    }

    for(j=0;j<num_of_group;j++)
    {
      // create incoming edge for each vertice i
      if ( g[i]->start_char == (g[j]->stop_char+1) )
         //  may not need to check (i != j) if all group have more than 2 char
      {
        gph->edge[j+1][i+1] = g[i]->group_weight;
      }

      // create outgoing edge for each vertice i (not need)
    }
  }
}

void f_print_graph(FILE *fp,GRAPH *gph)
{
  int i,j;

  fprintf(fp,"\n Printing graph infomation");
  fprintf(fp,"\n Number of vertice : %2d",gph->num_of_vertice);

  fprintf(fp,"\n\t vertices list");
  fprintf(fp,"\n\t   vertice[ 0] : dummy");
  for(i=1;i<gph->num_of_vertice-1;i++)
    fprintf(fp,"\n\t   vertice[%2d] : %s",i,gph->vertice[i]->str);
  fprintf(fp,"\n\t   vertice[%2d] : dummy",gph->num_of_vertice-1);

  fprintf(fp,"\n\t edges list");
  for(i=0;i<gph->num_of_vertice;i++)
    for(j=0;j<gph->num_of_vertice;j++)
    {
      if (gph->edge[i][j] != -1)
        fprintf(fp,"\n\t   edge from [%2d] to [%2d]  weight : %d",i,j,gph->edge[i][j]);
    }

}


void shortest_path(GRAPH *gph,int *length,int *path_length,int path[])
```

```
{
  // find shortest path in graph
  int i,j;
  int from_vertice[MAXGROUPINLINE];
  int current_length[MAXGROUPINLINE];
  int tmp_path[MAXGROUPINLINE];
  int p_length = 0;
  int bounded[MAXGROUPINLINE];
  int count;
  int updated;
  int min;
  int sl_min;

  for(i=1;i<gph->num_of_vertice;i++)
  {
    from_vertice[i] = i;
    current_length[i] = INTMAX;
    bounded[i] = 0;
  }
  current_length[0] = 0;
  bounded[0] = 1;
  count = 1;

  while(count<gph->num_of_vertice)
  {
    // update vertice that can reach by bounded vertice
    updated = 0;
    for(i=1;i<gph->num_of_vertice;i++)
    {
      for(j=0;j<gph->num_of_vertice;j++)
      {
        if ( (!bounded[i]) && bounded[j] &&
           (gph->edge[j][i] >= 0) && (i!=j) )
        {
          if ((gph->edge[j][i]+current_length[j]) < current_length[i])
          {
            current_length[i] = gph->edge[j][i]+current_length[j];
            from_vertice[i] = j;
          }
          updated = 1;
        }
      }
    }

    if (updated)
    {
      min = INTMAX;
      // select min
      for(i=1;i<gph->num_of_vertice;i++)
      {
        if ( (!bounded[i]) && (from_vertice[i] != i) && (current_length[i] < min) )
        {
          min =  current_length[i];
          sl_min = i;
        }
      }
      bounded[sl_min] = 1;
    }
    count++;
  }

  i = gph->num_of_vertice-1;
  tmp_path[p_length++] = gph->num_of_vertice-1;
  while(i>0)
  {
    tmp_path[p_length++] = from_vertice[i];
    i = from_vertice[i];
  }
  *path_length = p_length;
  *length  = current_length[gph->num_of_vertice-1];
  for(i=0;i<p_length;i++)
  {
    path[i] = tmp_path[p_length-i-1];
    //printf("\n vetice [%2d] reach by cost : %d ",path[i],current_length[path[i]]);
  }

}


void f_print_shortest_path(FILE *fp,GRAPH *gph,int *length,int *path_length,int path[])
```

```
{
  int i;

  fprintf(fp,"\n Printing shortest path");
  fprintf(fp,"\n\t vertices list");
  // not print first edge to dummy vertice and
  // not print last egde to dummy vertice
  for(i=1;i<*path_length-1;i++)
  {
    fprintf(fp,"\n\t   vertice[%2d] : %s",path[i],gph->vertice[path[i]]->str);
  }
  fprintf(fp,"\n Totol path length : %d",*length );
}



/*-------- trie ADT -----------*/
typedef struct trie_rec {
                          int trie_node_type;
                          void *ptr;
                          int count;
                        }TRIE_REC;
typedef struct trie_internal_node {
                                    TRIE_REC child[NUM_OF_CHAR];
                                  }TRIE_INTERNAL_NODE;

typedef struct trie_leaf_node {
                                int word_number;
                                int start_position; // start charactor of word
                                struct trie_leaf_node *next;
                               }TRIE_LEAF_NODE;

typedef struct trie_leaf_list {
                                TRIE_LEAF_NODE *head;
                                TRIE_LEAF_NODE *tail;
                                unsigned char have_complete_word;
                                // a flag turn on if have complete word in list
                                //
                               }TRIE_LEAF_LIST;

void create_trie_leaf_node(TRIE_LEAF_NODE **n,int word_num,int start_position)
{
  TRIE_LEAF_NODE *tmp;
  tmp = (TRIE_LEAF_NODE *)(malloc(sizeof(TRIE_LEAF_NODE)));
  tmp->word_number = word_num;
  tmp->start_position = start_position;
  tmp->next = NULL;
  *n = tmp;
  trie_leaf_node_counter++;
}

int init_trie_leaf_list(TRIE_LEAF_LIST *l)
{
  TRIE_LEAF_NODE *temp;
  // create list dummy node
  create_trie_leaf_node(&temp,0,0);
  l->head = temp;
  l->tail = temp;
  l->have_complete_word = 0;
  trie_leaf_list_counter++;
  return 1;
}

void free_trie_leaf_list(TRIE_LEAF_LIST *l)
{
  TRIE_LEAF_NODE *current,*tmp;
  current = l->head;
  while(current->next != NULL)
  {
    tmp = current->next;
    current->next = current->next->next;
    free(tmp);
    trie_leaf_node_counter--;
  }
  free(l->head); // free dummy node
  trie_leaf_node_counter--;
}

int empty_trie_leaf_list(TRIE_LEAF_LIST *l)
```

```
{
  if (l->head == l->tail)
    return 1;
  else
    return 0;
}


int insert_trie_leaf_list(TRIE_LEAF_LIST *l,TRIE_LEAF_NODE *n)
{
  // assumming  element insert to list is comming in order
  if (empty_trie_leaf_list(l))
  {
    l->head->next = n;
    l->tail = n;
    l->head->word_number++;//increase list size by 1
  }
  else
  {
    l->tail->next = n;
    l->tail = n;
    l->head->word_number++;//increase list size by 1
  }
  if (n->start_position == 0)
  {
    l->have_complete_word = 1;
  }
  return 1;
}


void f_print_trie_leaf_list(FILE *fp,TRIE_LEAF_LIST *l,MY_WORD *dictionary)
{
  TRIE_LEAF_NODE *current;
  MY_WORD *cw;
  current = l->head;
  while(current->next != NULL)
  {
    cw = dictionary+(current->next->word_number);
    fprintf(fp,"\n     [%5d] : %s",current->next->word_number,(char *)cw->w);
    current = current->next;
  }
}


void init_trie(CONFIG_VAR *cfg ,TRIE_REC **trie)
{
  *trie = (TRIE_REC *)(malloc(sizeof(TRIE_REC)));
  (*trie)->trie_node_type = LEAF_LIST;
  (*trie)->ptr = NULL;
  (*trie)->count = 0;

}

void free_trie(TRIE_REC *trie)
{ // must write it but later he he

  TRIE_INTERNAL_NODE *tmp_internal_node;
  TRIE_LEAF_LIST *tmp_leaf_list;
  int i;
  if (trie->ptr == NULL)
  {
  }
  else
  {
    if (trie->trie_node_type == LEAF_LIST)
    {
      tmp_leaf_list = (TRIE_LEAF_LIST *)(trie->ptr);
      free_trie_leaf_list(tmp_leaf_list);
      free(tmp_leaf_list);
      trie_leaf_list_counter--;
    }
    else
    {
      tmp_internal_node = (TRIE_INTERNAL_NODE *)(trie->ptr);
      // maintain all char
      for(i=0;i<NUM_OF_CHAR;i++)
      {
        if (tmp_internal_node->child[i].ptr != NULL)
        {
```

```
          free_trie(&(tmp_internal_node->child[i]));
        }
      }
      free(tmp_internal_node);
      trie_internal_node_counter--;
    }
  }
}


void create_trie_leaf_list(void **l)
{
  TRIE_LEAF_LIST *tmp;
  tmp = (TRIE_LEAF_LIST *)(malloc(sizeof(TRIE_LEAF_LIST)));
  init_trie_leaf_list(tmp);
  *l = (void *)(tmp);
}

void create_trie_internal_node(void **n)
{
  TRIE_INTERNAL_NODE *tmp;
  int i;

  tmp = (TRIE_INTERNAL_NODE *)(malloc(sizeof(TRIE_INTERNAL_NODE)));

  for(i=0;i<NUM_OF_CHAR;i++)
  {
    tmp->child[i].trie_node_type = LEAF_LIST;
    tmp->child[i].ptr = NULL;
    tmp->child[i].count = 0;
  }
  *n = (void *)(tmp);
  trie_internal_node_counter++;
}

void insert_trie_r(CONFIG_VAR *cfg,MY_WORD *dictionary,TRIE_REC *trie,int word_num,int
start_position,int position)
{
  TRIE_LEAF_LIST *tmp_leaf_list;
  TRIE_LEAF_NODE *tmp_leaf_node;
  TRIE_INTERNAL_NODE *tmp_internal_node;
  MY_WORD *w1,*w2;
  unsigned char c1,c2;
  int comp_res;
  char *str1,*str2;

  w1 = dictionary + word_num;
  c1 = *(w1->w + start_position + position);
  str1 = w1->w + start_position + position;
  trie->count++; // <== check about this line
  if ( (c1 == 0) ||
       (trie->ptr == NULL) )

  { // case :: terminate
    // it's enter this case if there is nothing in tries
    // or reach the end of word

    if (trie->ptr == NULL)
    { // case :: nothing in trie
      // creat leaf list
      create_trie_leaf_list(&(trie->ptr));
      trie->trie_node_type = LEAF_LIST;
      // create node and insert it to list
      create_trie_leaf_node(&tmp_leaf_node,word_num,start_position);
      insert_trie_leaf_list((TRIE_LEAF_LIST *)(trie->ptr),tmp_leaf_node);

    } // end case :: nothing in trie
    else
    { // case :: reach the end of word
      // (sth in trie and reach the end of word)

      //        printf("\n reach end of word [%d]:[%d]:[%d]",
      //                                  start_position + position,
      //                                  start_position,position);
      if ( trie->trie_node_type == LEAF_LIST )
      { // case :: leaf list
        // need split

        //printf("\n reach end of word [%d]:[%d]:[%d]",
        //                              start_position + position,
```

```
      //                              start_position,position);
      tmp_leaf_list = (TRIE_LEAF_LIST *)(trie->ptr);
      tmp_leaf_node = tmp_leaf_list->head->next;
      w2 = dictionary + tmp_leaf_node->word_number;
      c2 = *(w2->w + tmp_leaf_node->start_position + position);
      str2 = w2->w + tmp_leaf_node->start_position + position;
      comp_res = strcmp(str1,str2);
      if (comp_res != 0)
      { // case :: need split

        create_trie_internal_node(&(trie->ptr));
        trie->trie_node_type = INTERNAL_NODE;
        tmp_internal_node = (TRIE_INTERNAL_NODE *)(trie->ptr);

        // maintain old leaf list ## this case mean old leaf list have sth not
        //                           coz compres not equal to empty string
          tmp_internal_node->child[c2-cfg->start_char_ascii_no+1].trie_node_type =
          LEAF_LIST;
          tmp_internal_node->child[c2-cfg->start_char_ascii_no+1].ptr =
          (void *)(tmp_leaf_list);
          tmp_internal_node->child[c2-cfg->start_char_ascii_no+1].count =
          tmp_leaf_list->head->word_number;
        // end maintain old leaf list

        // maintain new word
          // create leaf list at 0 (terminate symbol) and add new word to it
          insert_trie_r(cfg,dictionary,
                        &(tmp_internal_node->child[0]),
                        word_num,
                        start_position,position);

        // end maintain new word
      } // end case :: need split
      else
      { // case :: not need split
        // create node and insert it to list
        create_trie_leaf_node(&tmp_leaf_node,word_num,start_position);
        insert_trie_leaf_list((TRIE_LEAF_LIST *)(trie->ptr),tmp_leaf_node);
      } // end case :: not need split
    } // end case :: leaf list
    else
    { // case :: internal node
      tmp_internal_node = (TRIE_INTERNAL_NODE *)(trie->ptr);
      insert_trie_r(cfg,dictionary,
                    &(tmp_internal_node->child[0]),
                    word_num,
                    start_position,position);
    } // end case:: internal node
  } // end case :: reach the end of word
} // end case :: terminate
else
{ // case :: reduction ( trie already have sth and it not reach the end
  // of word)
  if ( trie->trie_node_type == LEAF_LIST )
  { // case :: leaf list
    // need split
    tmp_leaf_list = (TRIE_LEAF_LIST *)(trie->ptr);
    tmp_leaf_node = tmp_leaf_list->head->next;
    w2 = dictionary + tmp_leaf_node->word_number;
    c2 = *(w2->w + tmp_leaf_node->start_position + position);
    str2 = w2->w + tmp_leaf_node->start_position + position;
    comp_res = strcmp(str1,str2);
    if (comp_res != 0)
    { // case :: need split
      // printf("\n XX :: %s | %s",str1,str2);
      create_trie_internal_node(&(trie->ptr));
      trie->trie_node_type = INTERNAL_NODE;
      tmp_internal_node = (TRIE_INTERNAL_NODE *)(trie->ptr);

      // maintain old leaf list
      if ( c2 == 0) // c2 = '\0' end of word
      { // case :: terminate symbol
        // move old list to terminate symbol
        tmp_internal_node->child[0].trie_node_type = LEAF_LIST;
        tmp_internal_node->child[0].ptr = (void *)(tmp_leaf_list);
        tmp_internal_node->child[0].count = tmp_leaf_list->head->word_number;
      } // end case :: terminate symbol
      else
      { // case :: other symbol
        // move old list to new place
```

```
          tmp_internal_node->child[c2-cfg->start_char_ascii_no+1].trie_node_type =
          LEAF_LIST;
          tmp_internal_node->child[c2-cfg->start_char_ascii_no+1].ptr =
          (void *)(tmp_leaf_list);
          tmp_internal_node->child[c2-cfg->start_char_ascii_no+1].count =
          tmp_leaf_list->head->word_number;
        } // end case :: other symbol
        // end maintain old leaf list

        // maintain new word
          insert_trie_r(cfg,dictionary,
                        &(tmp_internal_node->child[c1-cfg->start_char_ascii_no+1]),
                        word_num,
                        start_position,position+1);
        // end maintain new word

      } // end case :: need split
      else
      { // case :: not need split
        create_trie_leaf_node(&tmp_leaf_node,word_num,start_position);
        insert_trie_leaf_list((TRIE_LEAF_LIST *)(trie->ptr),tmp_leaf_node);
      } // end case :: not need split

    } // end case :: leaf list
    else
    { // case :: internal node
        tmp_internal_node = (TRIE_INTERNAL_NODE *)(trie->ptr);
        insert_trie_r(cfg,dictionary,
                      &(tmp_internal_node->child[c1-cfg->start_char_ascii_no+1]),
                      word_num,
                      start_position,position+1);

    } // end case:: internal node

  } // end case:: reduction

}

void insert_trie(CONFIG_VAR *cfg,MY_WORD *dictionary,TRIE_REC *trie,int word_num,int
position)
{
  MY_WORD *current_word;
  char *c;

  current_word = dictionary+word_num;
  c = current_word->w + position;
  insert_trie_r(cfg,dictionary,trie,word_num,position,0);
}

void insert_word_to_trie(CONFIG_VAR *cfg,MY_WORD *dictionary,TRIE_REC *trie,int
word_num)
{
  // insert word and it's substring in to trie
  int i;
  MY_WORD *current_word;

  current_word = dictionary+word_num;
  // insert word
  insert_trie(cfg,dictionary,trie,word_num,0);
  // insert word substring
  for(i=1;i<current_word->wlength;i++)
  {
    insert_trie(cfg,dictionary,trie,word_num,i);
  }
}

void make_trie(CONFIG_VAR *cfg,MY_WORD *dictionary,TRIE_REC **trie)
{
  int i;
  init_trie(cfg,trie);
  //
  if ( (*trie)->ptr == NULL )
    printf("\n initailize is correct");

  for(i=0;i<cfg->total_word_in_dict; i++)
  //for(i=3;i<5; i++)
  {
    insert_word_to_trie(cfg,dictionary,*trie,i);
    //printf("\n%d",i);
  }
```

```
}

void f_print_trie(FILE *fp,CONFIG_VAR *cfg,MY_WORD *dictionary,TRIE_REC *trie,char *s)
{
  char buffer[MAXWORDLENGTH];
  TRIE_LEAF_NODE *tmp_leaf_node;
  TRIE_INTERNAL_NODE *tmp_internal_node;
  TRIE_LEAF_LIST *tmp_leaf_list;
  int i;
  if (trie->ptr == NULL)
  {
    //fprintf(fp,"*");
  }
  else
  {
    if (trie->trie_node_type == LEAF_LIST)
    {
      tmp_leaf_list = (TRIE_LEAF_LIST *)(trie->ptr);
      tmp_leaf_node = tmp_leaf_list->head->next;
      fprintf(fp,"\n leaf list");
      f_print_trie_leaf_list(fp,tmp_leaf_list,dictionary);
    }
    else
    {
      tmp_internal_node = (TRIE_INTERNAL_NODE *)(trie->ptr);
      // maintain terminate char
      if (tmp_internal_node->child[0].ptr != NULL)
      {
        fprintf(fp,"\n internal [%s]",s);
      }
      // maintain all char
      for(i=0;i<NUM_OF_CHAR;i++)
      {
        if (tmp_internal_node->child[i].ptr != NULL)
        {
          sprintf(buffer,"%s%c",s,cfg->start_char_ascii_no+i-1);
          f_print_trie(fp,cfg,dictionary,&(tmp_internal_node->child[i]),buffer);
        }
      }
    }
  }
}

int search_trie(CONFIG_VAR *cfg,MY_WORD *dictionary,TRIE_REC *trie,char *str,int
position)
{ // return 1 if found str in trie return 0 if not
  int ret = 0;
  TRIE_LEAF_LIST *tmp_leaf_list;
  TRIE_LEAF_NODE *tmp_leaf_node;
  TRIE_INTERNAL_NODE *tmp_internal_node;
  MY_WORD *cw;
  int comp_res;
  char *c,*cstr;
  unsigned char uc;

  cstr = str + position;
  if (trie->ptr  == NULL)
  {
  }
  else
  {
    if (trie->trie_node_type == LEAF_LIST)
    {
      tmp_leaf_list = (TRIE_LEAF_LIST  *)(trie->ptr);
      tmp_leaf_node = tmp_leaf_list->head->next;
      cw = dictionary+tmp_leaf_node->word_number;
      c = cw->w + tmp_leaf_node->start_position + position;
      comp_res = strcmp(c,cstr);
      if (comp_res == 0)
      {
        ret = 1;
      }
    }
    else
    {
      tmp_internal_node = (TRIE_INTERNAL_NODE *)(trie->ptr);
      if (*cstr == '\0')
      {
        ret = search_trie(cfg,dictionary,&(tmp_internal_node->child[0]),str,position);
```

```
      }
      else
      {
        uc = *cstr;
        ret = search_trie(cfg,dictionary,&(tmp_internal_node->child[uc-cfg->
start_char_ascii_no+1]),str,position+1);
      }
    }
  }

  return(ret);
}

int comp(char *str,char *pat)
{
  // return 1 if pat is same as head of str
  int ret = 1;
  int i=0;
  int miss = 0;
  char c,s;
  c = *pat;
  // printf("\n [%s] :: [%s] ",str,pat);
  while ( (!miss) && (c != '\0') )
  {
    c = *(pat+i);
    s = *(str+i);
    // printf("\n [%c] | [%c]",c,s);
    if (s != c)
    {
      ret = 0;
      miss = 1;
    }
    i++;
    c = *(pat+i);
  }

  return(ret);
}

int get_num_trie(CONFIG_VAR *cfg,MY_WORD *dictionary,TRIE_REC *trie,char *str,
                 int position)
{ // return Uniqueness of str
  int ret = 0;
  TRIE_LEAF_LIST *tmp_leaf_list;
  TRIE_LEAF_NODE *tmp_leaf_node;
  TRIE_INTERNAL_NODE *tmp_internal_node;
  MY_WORD *cw;
  int comp_res;
  char *c,*cstr;
  unsigned char uc;

  cstr = str + position;
  uc   = (unsigned char)(*cstr);
  if (*cstr == '\0')
  {
    ret = trie->count;
  }
  else
  {
    if (trie->ptr  == NULL)
    {
      ret = 0; // return 0 but use -1 while debug
    }
    else
    {
      if (trie->trie_node_type == LEAF_LIST)
      {
        tmp_leaf_list = (TRIE_LEAF_LIST  *)(trie->ptr);
        tmp_leaf_node = tmp_leaf_list->head->next;
        cw = dictionary+tmp_leaf_node->word_number;
        c = cw->w + tmp_leaf_node->start_position + position;
        // must not use strcmp it'll produce wrong result

        comp_res = comp(c,cstr);
        if (comp_res == 1)
        {
          ret = tmp_leaf_list->head->word_number;
        }
        else
```

```
        {
          ret = 0; // return 0 but use -2 while debug
        }
      }
      else
      {
        tmp_internal_node = (TRIE_INTERNAL_NODE *)(trie->ptr);
        ret = get_num_trie(cfg,dictionary,
                           &(tmp_internal_node->child[uc-cfg->start_char_ascii_no+1]),
                           str,position+1);
      }
    }
  }
  return(ret);
}

TRIE_REC *get_trie(CONFIG_VAR *cfg,MY_WORD *dictionary,TRIE_REC *trie,char *str,int
position)
{ // return position in trie found str
  TRIE_REC *ret = NULL;
  TRIE_LEAF_LIST *tmp_leaf_list;
  TRIE_LEAF_NODE *tmp_leaf_node;
  TRIE_INTERNAL_NODE *tmp_internal_node;
  MY_WORD *cw;
  int comp_res;
  char *c,*cstr;
  unsigned char uc;

  cstr = str + position;
  uc   = (unsigned char)(*cstr);
  if (*cstr == '\0')
  {
    ret = trie;
  }
  else
  {
    if (trie->ptr  == NULL)
    {
      ret = NULL; // return 0 but use -1 while debug
    }
    else
    {
      if (trie->trie_node_type == LEAF_LIST)
      {
        tmp_leaf_list = (TRIE_LEAF_LIST  *)(trie->ptr);
        tmp_leaf_node = tmp_leaf_list->head->next;
        cw = dictionary+tmp_leaf_node->word_number;
        c = cw->w + tmp_leaf_node->start_position + position;
        // must not use strcmp it'll produce wrong result

        comp_res = comp(c,cstr);
        if (comp_res == 1)
        {
          ret = trie;
        }
        else
        {
          ret = NULL; // return 0 but use -2 while debug
        }
      }
      else
      {
        tmp_internal_node = (TRIE_INTERNAL_NODE *)(trie->ptr);
        ret = get_trie(cfg,dictionary,
                       &(tmp_internal_node->child[uc-cfg->start_char_ascii_no+1]),
                       str,position+1);
      }
    }
  }
  return(ret);
}


int complete_word(CONFIG_VAR *cfg,MY_WORD *dictionary,TRIE_REC *trie,char *str,int
position)
{ // this function return 1 if str can be a complete word else return 0
  int ret = 0;
  TRIE_LEAF_LIST *tmp_leaf_list;
  TRIE_LEAF_NODE *tmp_leaf_node;
  TRIE_INTERNAL_NODE *tmp_internal_node;
```

```
  MY_WORD *cw;
  int comp_res;
  char *c,*cstr;
  unsigned char uc;

  cstr = str + position;
  if (trie->ptr  == NULL)
  {
  }
  else
  {
    if (trie->trie_node_type == LEAF_LIST)
    {
      tmp_leaf_list = (TRIE_LEAF_LIST  *)(trie->ptr);
      tmp_leaf_node = tmp_leaf_list->head->next;
      cw = dictionary+tmp_leaf_node->word_number;
      c = cw->w + tmp_leaf_node->start_position + position;
      comp_res = strcmp(c,cstr);
      if (comp_res == 0)
      {
        if(tmp_leaf_list->have_complete_word == 1)
        {
          ret = 1;
        }
      }
    }
    else
    {
      tmp_internal_node = (TRIE_INTERNAL_NODE *)(trie->ptr);
      if (*cstr == '\0')
      {
        ret = complete_word(cfg,dictionary,
                            &(tmp_internal_node->child[0]),str,position);
      }
      else
      {
        uc = *cstr;
        ret = complete_word(cfg,dictionary,
                            &(tmp_internal_node->child[uc-cfg->start_char_ascii_no+1]),
                            str,position+1);
      }
    }
  }

  return(ret);
}


/*-------- candidate ADT -----------*/
typedef struct candidate_node {
                                char str[MAXWORDLENGTH];
                                int completeness; // able to be a complete word
                                struct candidate_node *next;
                              }CANDIDATE_NODE;

typedef struct candidate_list {
                                CANDIDATE_NODE *head;
                                CANDIDATE_NODE *tail;
                                int num_of_candidate;
                              }CANDIDATE_LIST;

typedef struct candidate  {
                            CANDIDATE_LIST insertion;
                            CANDIDATE_LIST deletetion;
                            CANDIDATE_LIST substitutetion;
                            CANDIDATE_LIST transposition;
                          }CANDIDATE;


void create_candidate_node(char *str,int completeness,CANDIDATE_NODE **n)
{
  CANDIDATE_NODE *temp;
  temp = (CANDIDATE_NODE *)(malloc(sizeof(CANDIDATE_NODE)));
  strcpy(temp->str,str);
  temp->completeness = completeness;
  temp->next = NULL;
  *n = temp;
  number_of_candidate_nodes_used++;
}
```

```
int init_candidate_list(CANDIDATE_LIST *l)
{
  CANDIDATE_NODE *temp;
  char str[MAXWORDLENGTH];
  str[0] = '\0';
  create_candidate_node(str,0,&temp);
  l->head = temp;
  l->tail = temp;
  l->num_of_candidate = 0;
  return 1;
}

void free_candidate_list(CANDIDATE_LIST *l)
{
  CANDIDATE_NODE *current,*tmp;
  current = l->head;
  while(current->next != NULL)
  {
    tmp = current->next;
    current->next = current->next->next;
    free(tmp);
    number_of_candidate_nodes_used--;
  }
  free(l->head); // free dummy node
  number_of_candidate_nodes_used--;
}

int empty_candidate_list(CANDIDATE_LIST *l)
{
  if (l->head == l->tail)
    return 1;
  else
    return 0;
}


int insert_candidate_list(CANDIDATE_LIST *l,CANDIDATE_NODE *n)
{
  // assumming  element insert to list is comming in order
  if (empty_candidate_list(l))
  {
    l->head->next = n;
    l->tail = n;
    l->num_of_candidate++;//increase list size by 1
  }
  else
  {
    l->tail->next = n;
    l->tail = n;
    l->num_of_candidate++;//increase list size by 1
  }
  return 1;
}

void f_print_candidate_list(FILE *fp,CANDIDATE_LIST *l)
{
  CANDIDATE_NODE *current;
  current = l->head;
  while(current->next != NULL)
  {
    fprintf(fp,"\n\t%s",current->next->str);
    if (current->next->completeness == 1)
      fprintf(fp,"[*]");
    current = current->next;
  }
}


void init_candidate(CANDIDATE *c)
{
  init_candidate_list(&(c->insertion));
  init_candidate_list(&(c->deletetion));
  init_candidate_list(&(c->substitutetion));
  init_candidate_list(&(c->transposition));
}



void free_candidate(CANDIDATE *c)
```

```
{
  free_candidate_list(&(c->insertion));
  free_candidate_list(&(c->deletetion));
  free_candidate_list(&(c->substitutetion));
  free_candidate_list(&(c->transposition));
}


void insertion_error_approx_match(CONFIG_VAR *cfg,MY_WORD *dictionary,
                                  TRIE_REC *trie,char *original_str,CANDIDATE_LIST *l)
{
  char buffer[MAXWORDLENGTH],buffer_fixed[MAXWORDLENGTH];
  char h[MAXWORDLENGTH],t[MAXWORDLENGTH];
  TRIE_REC *tmp;
  int i,j,error_point,length;
  CANDIDATE_NODE *tmp_candidate_node;
  int completeness;

  strcpy(buffer,original_str);
  length = strlen(buffer);

    for(error_point=0;error_point<length;error_point++)
    {
      //copy head part ( part before error )
      for(i=0;i<error_point;i++)
      {
        h[i] = buffer[i];
      }
      h[i] = '\0';

      j = 0;
      for(i=error_point+1;i<length;i++)
      {
        t[j] = buffer[i];
        j++;
      }
      t[j] = '\0';

      // search trie
      sprintf(buffer_fixed,"%s%s",h,t);

      tmp = get_trie(cfg,dictionary,trie,buffer_fixed,0);
      if (tmp == NULL)
      {
        // do nothing coz this sequence of char after fixing is not valid
        //fprintf(fp,"  [invalid sequence of char]");
      }
      else
      {
        if (tmp->count == 0)
        {
          // do nothing coz this sequence of char after fixing is not valid
          //fprintf(fp,"  [invalid sequence of char]");
        }
        else
        {
          // add word to list
          completeness = complete_word(cfg,dictionary,trie,buffer_fixed,0);
          create_candidate_node(buffer_fixed,completeness,&tmp_candidate_node);
          insert_candidate_list(l,tmp_candidate_node);
        }
      }
    }
}

void deletetion_error_approx_match(CONFIG_VAR *cfg,MY_WORD *dictionary,
                                   TRIE_REC *trie,char *original_str,CANDIDATE_LIST *l)
{
  char buffer[MAXWORDLENGTH],buffer_fixed[MAXWORDLENGTH];
  char h[MAXWORDLENGTH],t[MAXWORDLENGTH];
  TRIE_REC *tmp;
  int i,j,k,error_point,length;
  CANDIDATE_NODE *tmp_candidate_node;
  int completeness;


  strcpy(buffer,original_str);
  length = strlen(buffer);

    for(error_point=0;error_point<=length;error_point++)
```

```
    {
      //copy head part ( part before error )
      for(i=0;i<error_point;i++)
      {
        h[i] = buffer[i];
      }
      h[i] = '\0';

      j = 0;
      for(i=error_point;i<length;i++)
      {
        t[j] = buffer[i];
        j++;
      }
      t[j] = '\0';

      for(k=cfg->start_char_ascii_no;k<=cfg->last_char_ascii_no;k++)
      {
        // search trie
        sprintf(buffer_fixed,"%s%c%s",h,(char)(k),t);

        tmp = get_trie(cfg,dictionary,trie,buffer_fixed,0);
        if (tmp == NULL)
        {
          // do nothing coz this sequence of char after fixing is not valid
          //fprintf(fp,"  [invalid sequence of char]");
        }
        else
        {
          if (tmp->count == 0)
          {
            // do nothing coz this sequence of char after fixing is not valid
            //fprintf(fp,"  [invalid sequence of char]");
          }
          else
          {
            // add word to list
            completeness = complete_word(cfg,dictionary,trie,buffer_fixed,0);
            create_candidate_node(buffer_fixed,completeness,&tmp_candidate_node);
            insert_candidate_list(l,tmp_candidate_node);
            //f_print_trie(fp,cfg,dictionary,tmp,buffer_fixed );
          }
        }
      }
    }
}

void substitutetion_error_approx_match(CONFIG_VAR *cfg,MY_WORD *dictionary,
                                       TRIE_REC *trie,char *original_str,
                                       CANDIDATE_LIST *l)
{
  char buffer[MAXWORDLENGTH],buffer_fixed[MAXWORDLENGTH];
  char h[MAXWORDLENGTH],t[MAXWORDLENGTH];
  TRIE_REC *tmp;
  int i,j,k,error_point,length;
  CANDIDATE_NODE *tmp_candidate_node;
  int completeness;


  strcpy(buffer,original_str);
  length = strlen(buffer);

    for(error_point=0;error_point<length;error_point++)
    {
      //copy head part ( part before error )
      for(i=0;i<error_point;i++)
      {
        h[i] = buffer[i];
      }
      h[i] = '\0';

      j = 0;
      for(i=error_point+1;i<length;i++)
      {
        t[j] = buffer[i];
        j++;
      }
      t[j] = '\0';

      for(k=cfg->start_char_ascii_no;k<=cfg->last_char_ascii_no;k++)
```

```
      {
        // search trie
        sprintf(buffer_fixed,"%s%c%s",h,(char)(k),t);

        tmp = get_trie(cfg,dictionary,trie,buffer_fixed,0);
        if (tmp == NULL)
        {
          // do nothing coz this sequence of char after fixing is not valid
          //fprintf(fp,"  [invalid sequence of char]");
        }
        else
        {
          if (tmp->count == 0)
          {
            // do nothing coz this sequence of char after fixing is not valid
            //fprintf(fp,"  [invalid sequence of char]");
          }
          else
          {
            // add word to list
            completeness = complete_word(cfg,dictionary,trie,buffer_fixed,0);
            create_candidate_node(buffer_fixed,completeness,&tmp_candidate_node);
            insert_candidate_list(l,tmp_candidate_node);
            //f_print_trie(fp,cfg,dictionary,tmp,buffer_fixed );
          }
        }
      }
    }
}

void transposition_error_approx_match(CONFIG_VAR *cfg,MY_WORD *dictionary,
                                      TRIE_REC *trie,char *original_str,
                                      CANDIDATE_LIST *l)
{ // substitute position i and i+1
  char buffer[MAXWORDLENGTH],buffer_fixed[MAXWORDLENGTH];
  char h[MAXWORDLENGTH],t[MAXWORDLENGTH];
  TRIE_REC *tmp;
  int i,j,error_point,length;
  CANDIDATE_NODE *tmp_candidate_node;
  int completeness;

  strcpy(buffer,original_str);
  length = strlen(buffer);

    for(error_point=0;(error_point+1)<length;error_point++)
    {
      //copy head part ( part before error )
      for(i=0;i<error_point;i++)
      {
        h[i] = buffer[i];
      }
      h[i] = '\0';

      j = 0;
      for(i=error_point+2;i<length;i++)
      {
        t[j] = buffer[i];
        j++;
      }
      t[j] = '\0';

      // search trie
      sprintf(buffer_fixed,"%s%c%c%s",h,buffer[error_point+1],buffer[error_point],t);

      tmp = get_trie(cfg,dictionary,trie,buffer_fixed,0);
      if (tmp == NULL)
      {
        // do nothing coz this sequence of char after fixing is not valid
        //fprintf(fp,"  [invalid sequence of char]");
      }
      else
      {
        if (tmp->count == 0)
        {
          // do nothing coz this sequence of char after fixing is not valid
          //fprintf(fp,"  [invalid sequence of char]");
        }
        else
        {
            // add word to list
```

```
            completeness = complete_word(cfg,dictionary,trie,buffer_fixed,0);
            create_candidate_node(buffer_fixed,completeness,&tmp_candidate_node);
            insert_candidate_list(l,tmp_candidate_node);
            //f_print_trie(fp,cfg,dictionary,tmp,buffer_fixed );
          }
        }
      }
}


void create_candidate(CONFIG_VAR *cfg,MY_WORD *dictionary,TRIE_REC *trie,char
*str,CANDIDATE *can)
{
  char buffer[MAXWORDLENGTH];
  int length;

  strcpy(buffer,str);
    length = strlen(buffer);

    // approx for insertion error
    if (length >= cfg->min_insertion_error/*4*/)
    {
      insertion_error_approx_match(cfg,dictionary,trie,buffer,&(can->insertion));
    }

    // approx for deletetion error
    if (length >= cfg->min_deletetion_error/*3*/ )
    {
      deletetion_error_approx_match(cfg,dictionary,trie,buffer,&(can->deletetion));
    }

    // approx for substitutetion error
    if (length >= cfg->min_substitutetion_error/*4*/)
    {
      substitutetion_error_approx_match(cfg,dictionary,trie,buffer,
                                        &(can->substitutetion));
    }

    // approx for transposition error
    if (length >= cfg->min_transposition_error/*3*/)
    {
      transposition_error_approx_match(cfg,dictionary,trie,buffer,
                                       &(can->transposition));
    }
}



/*-------- index ADT -----------*/
typedef struct index_node {
                            char str[MAXWORDLENGTH];
                            int  start_position;
                            int  stop_position;
                            int  type;
                            //   0   original word
                            //   1   correct insertion error
                            //   2   correct deletetion error
                            //   3   correct substitutetion error
                            //   4   correct transposition error
                            struct index_node *next;
                           }INDEX_NODE;

typedef struct index_list {
                            INDEX_NODE *head;
                            INDEX_NODE *tail;
                            int number_of_words;
                           }INDEX_LIST;

typedef struct index  {
                         INDEX_LIST base_index;
                         INDEX_LIST additional_index;
                       }INDEX;


void create_index_node(char *str,int start_position,int stop_position,int
type,INDEX_NODE **n)
{
  INDEX_NODE *temp;
  temp = (INDEX_NODE *)(malloc(sizeof(INDEX_NODE)));
```

```
  strcpy(temp->str,str);
  temp->start_position = start_position;
  temp->stop_position = stop_position;
  temp->type = type;
  temp->next = NULL;
  *n = temp;
  number_of_index_nodes_used++;
}

int init_index_list(INDEX_LIST *l)
{
  INDEX_NODE *temp;
  char str[MAXWORDLENGTH];
  str[0] = '\0';
  create_index_node(str,0,0,ORIGINAL,&temp);
  l->head = temp;
  l->tail = temp;
  l->number_of_words = 0;
  return 1;
}

void free_index_list(INDEX_LIST *l)
{
  INDEX_NODE *current,*tmp;
  current = l->head;
  while(current->next != NULL)
  {
    tmp = current->next;
    current->next = current->next->next;
    free(tmp);
    number_of_index_nodes_used--;
  }
  free(l->head); // free dummy node
  number_of_index_nodes_used--;
}

int empty_index_list(INDEX_LIST *l)
{
  if (l->head == l->tail)
    return 1;
  else
    return 0;
}


int insert_index_list(INDEX_LIST *l,INDEX_NODE *n)
{
  int compres;
  INDEX_NODE *current;
  int inserted = 0;
  int same;

  current = l->head;
  while ( (current->next != NULL) && (!inserted) )
  {
    compres = strcmp(n->str,current->next->str);
    if (compres == 0)
    {
      // must check start and stop position  and type if same no need to add
      // if not add it
      // must search all that have same str before add
      same = 0;
      while ( (!same) && (compres == 0) && (current->next != NULL) )
      {
        if ( (current->next->start_position == n->start_position) &&
             (current->next->stop_position == n->stop_position) &&
             (current->next->type == n->type) )
        {
          free(n);
          number_of_index_nodes_used--;
          same = 1;
          inserted = 1;
        }

        else
        {
          current = current->next;
          if ( current->next != NULL )
          {
            compres = strcmp(n->str,current->next->str);
```

```
          }
        }
      }
      if (!same)
      {
        n->next = current->next;
        current->next = n;
        l->number_of_words++;//increase list size by 1
        inserted = 1;
      }
    }
    else
    {
      if (compres < 0)
      {
        n->next = current->next;
        current->next = n;
        l->number_of_words++;//increase list size by 1
        inserted = 1;
      }
      else
      {
        current = current->next;
      }
    }
  }
  // insert at tail
  if (!inserted)
  {
    current->next = n;
    l->tail = n;
    l->number_of_words++;//increase list size by 1
  }
  return 1;
}



void init_index(INDEX *ind)
{
  init_index_list(&(ind->base_index));
  init_index_list(&(ind->additional_index));
}

void free_index(INDEX *ind)
{
  free_index_list(&(ind->base_index));
  free_index_list(&(ind->additional_index));
}

void create_base_index(FILE *fp,INDEX_LIST *l,CONFIG_VAR *cfg,
                       MY_WORD *dictionary,
                       TRIE_REC *trie,GROUP *g[],
                       int g_c,int e_g_c,
                       GRAPH *gph,int path[],int path_length,int length)
{
  int i,j,k,b1,b2,wf,wb,hwf,hwb;
  int minimum_length = cfg->min_original_accept_length;
  INDEX_NODE *tmp_index_node;
  int CC = cfg->CC;
  int AA = cfg->AA;
  int BB = cfg->BB;
  int flag = 0;

  fprintf(fp,"\n additional words \n");
  for(i=1;i<path_length-1;i++)
  {
    // add current node to index if it complete word
    if ( strlen(gph->vertice[path[i]]->str) >= (unsigned int)(minimum_length) )
    {
      if (complete_word(cfg,dictionary,trie,gph->vertice[path[i]]->str,0) )
      {
        create_index_node(gph->vertice[path[i]]->str,
                          gph->vertice[path[i]]->start_char,
                          gph->vertice[path[i]]->stop_char,
                          0,&tmp_index_node);
        insert_index_list(l,tmp_index_node);
      }
    }
```

```
// looking for ambiguous
if (i+1 < path_length-1)
{
  for(j=0;j<e_g_c;j++)
  {
    if ( strlen(g[j]->str) >= (unsigned int)(minimum_length) )
    {
      if ( (g[j]->start_char >= gph->vertice[path[i]]->start_char ) &&
           (g[j]->stop_char <= gph->vertice[path[i+1]]->stop_char) &&
           !( (g[j]->start_char == gph->vertice[path[i]]->start_char) &&
              (g[j]->stop_char == gph->vertice[path[i]]->stop_char) ) &&
           !( (g[j]->start_char == gph->vertice[path[i+1]]->start_char) &&
              (g[j]->stop_char == gph->vertice[path[i+1]]->stop_char) )  )

      {

        if (complete_word(cfg,dictionary,trie,g[j]->str,0) )
        {
          b1 = 1;
          hwf = 0;
          if (g[j]->start_char > gph->vertice[path[i]]->start_char)
          { // search for part come before g[j]
            b1 = 0;
            for(k=0;k<e_g_c;k++)
            {
              if ( (g[k]->start_char == gph->vertice[path[i]]->start_char) &&
                   (g[k]->stop_char == g[j]->start_char-1) &&
                   (g[j]->stop_char != gph->vertice[path[i]]->stop_char) )


              {
                if ( strlen(g[k]->str) >= (unsigned int)(minimum_length) )
                {
                  if (complete_word(cfg,dictionary,trie,g[k]->str,0) )
                  {
                    b1 = 1;
                    wf = k;
                    hwf = 1;
                  }
                }
              }
            }
          }
          b2 =1;
          hwb = 0;
          if (g[j]->stop_char < gph->vertice[path[i+1]]->stop_char)
          { // search for part come after g[j]
            b2 = 0;
            for(k=0;k<e_g_c;k++)
            {
              if ( (g[k]->start_char == g[j]->stop_char+1 ) &&
                   (g[k]->stop_char == gph->vertice[path[i+1]]->stop_char) &&
                   (g[j]->start_char != gph->vertice[path[i+1]]->start_char) )
              {
                if ( strlen(g[k]->str) >= (unsigned int)(minimum_length) )
                {
                  if (complete_word(cfg,dictionary,trie,g[k]->str,0) )
                  {
                    b2 = 1;
                    wb = k;
                    hwb = 1;
                  }
                }
              }
            }
          }

          if (b1 && b2)
          {
            create_index_node(g[j]->str,g[j]->start_char,g[j]->stop_char,
                              0,&tmp_index_node);
            insert_index_list(l,tmp_index_node);

            if (hwf)
            {
              create_index_node(g[wf]->str,g[wf]->start_char,g[wf]->stop_char,
                                0,&tmp_index_node);
              insert_index_list(l,tmp_index_node);
            }
```

```
                if (hwb)
                {
                  create_index_node(g[wb]->str,g[wb]->start_char,g[wb]->stop_char,
                                    0,&tmp_index_node);
                  insert_index_list(1,tmp_index_node);
                }
              }
            }
          }
        }
      }
    }
  }

  f_print_index_list(fp,1);
}

void remove_duplicate(INDEX *ind)
{
  INDEX_NODE *current_base_index,*current_additional_index;
  INDEX_LIST *additional_index;
  INDEX_LIST *base_index;
  int compres;
  INDEX_NODE *tmp;

  additional_index = &(ind->additional_index);
  base_index = &(ind->base_index);

  current_base_index = base_index->head;
  current_additional_index = additional_index->head;

  while( (current_base_index->next != NULL) &&
         (current_additional_index->next != NULL)  )
  {
    compres = strcmp(current_base_index->next->str,
                     current_additional_index->next->str);
    if (compres < 0)
    {
      current_base_index = current_base_index->next;
    }
    else
    {
      if (compres > 0)
      {
        current_additional_index = current_additional_index->next;
      }
      else
      {
        // remove node from additional index list
        tmp = current_additional_index->next;
        current_additional_index->next = tmp->next;
        if (additional_index->tail == tmp)
        {
          additional_index->tail = current_additional_index;
        }
        free(tmp);
        number_of_index_nodes_used--;
        additional_index->number_of_words--;//decrease list size by 1
      }
    }

  }
}


/*----- function using in separator process -----*/

int is_in_dictionary(MY_WORD *dictionary,int total_word,
                          char input_str[])
{
// This function return entry number where input_str locate in
// dictionary if found  or return -1 if not found
// total_word number of words in dictionary
// return entry number where str locate in dictionary if found[0-totalword-1]
// or return -1 if not found
  int low=0,high=total_word-1,mid;
  int cmpres;//string compare result
  char *cwd;//current word_in dictionary to compare
  int found = 0;
```

```
  int ret = -1;

  while( (low<=high) && (!found) )
  {
    mid = (low+high)/2;
    cwd = (char *)((dictionary+mid)->w);
    cmpres = strcmp(input_str,cwd);
    if (cmpres < 0)
    {
      high=mid-1;
    }
    else
    {
      if (cmpres > 0)
      {
        low=mid+1;
      }
      else // case cmpres == 0
      {
        found = 1;
        ret = mid;
      }
    }
  }
  return (ret);
}

void longest_match(CONFIG_VAR *cfg,MY_WORD *dictionary,
                   int total_word,int position,
                   int *last_match_position,
                   char input_str[],char output_str[])
{
// This function will find the longest match word start from position
// i in input_str[](T) the parameter last_match_position will set to
// j that make f_{i,j} longest word
  int i;
  int last_match;
  int input_str_length = strlen(input_str);
  char current_str[MAXWORDLENGTH];
  int entry_number;
  MY_WORD *cw;

  i=position;
  current_str[i-position] = input_str[i];
  current_str[i-position+1] = '\0';
  last_match = i;
  for(i=position+1;(i<input_str_length)&&( (i-position)< cfg->beta_factor );i++)
  {
    current_str[i-position] = input_str[i];
    current_str[i-position+1] = '\0';
    entry_number = is_in_dictionary(dictionary,total_word,current_str);
    cw = dictionary+entry_number;
    if ( (entry_number >=0) && (entry_number < (total_word-1)) )
    {

      if ( position == 0 )
      {
        if (i == (input_str_length-1) )
        { // accept stand alone word only
          if ( cw->flag[stand_alone] )
          {
            last_match = i;
          }
        }
        else
        { // accept stand alone or prefix
          if ( cw->flag[stand_alone] ||
               cw->flag[prefix] )
          {
            last_match = i;
          }
        }
      }
      else
      {
        if (i == (input_str_length-1) )
        { // accept stand alone or sufffix word
          if ( cw->flag[stand_alone] ||
               cw->flag[suffix] )
          {
```

```
            last_match = i;
          }
        }
        else
        { // accept all kind of word
            last_match = i;
        }
      }
    }
  }
  current_str[last_match-position+1] = '\0';
  strcpy(output_str,current_str);
  *last_match_position = last_match;

}

void sep(CONFIG_VAR *cfg,MY_WORD *dictionary,void *ngram_table,
          void *memspace,TRIE_REC *trie,GROUP *g[],
          int  *g_c,int  *e_g_c,char buffer[])
{
         char longest_match_str[MAXLINELENGTH][MAXWORDLENGTH];
         int  line_length;
         int  i,j,k,l;
         // last_match_position : keep the position of last char
         // in sentence for each longest word
         int  last_match_position[MAXLINELENGTH];
         char tmp_str[MAXWORDLENGTH];
         int  is_part_of_former_group;
         int have_incoming_edge;
         int have_outgoing_edge;
         int  group_count;
         int  extended_group_count;
         int  minimum_length = cfg->min_original_accept_length;
         SET *tmp_set;

         line_length = strlen(buffer);
        for(i=0;i<line_length;i++)
        {
            longest_match(cfg,dictionary,cfg->total_word_in_dict,
                          i,last_match_position+i,
                          buffer,longest_match_str[i]);
        }

        group_count = 0;
        for(i=0;i<line_length;i++)
        {
          is_part_of_former_group = 0;
          for(j=0;j<group_count;j++)
          {
            // is completely substring of group
            if ( (i >= g[j]->start_char ) &&
                 (last_match_position[i] <= g[j]->stop_char ) )
            {
              is_part_of_former_group = 1;
            }
          }
          if (!is_part_of_former_group)
          {
            g[group_count] = (GROUP *)(malloc(sizeof(GROUP)));
            init_group(g[group_count]);
            setup_group(g[group_count],longest_match_str[i],
                        i,last_match_position[i]);
            group_count++;
          }
        } // end : for(i=0;i<line_length;i++)

      // create extended group : inside word, overlap word
      //                       : completely inside word

        extended_group_count = group_count;

        // create completely inside word
        for(i=0;i<line_length;i++)
        {
          is_part_of_former_group = 0;
          for(j=0;j<group_count;j++)
          {
            // is completely substring of group
            if ( (i >= g[j]->start_char ) &&
                 (last_match_position[i] <= g[j]->stop_char ) &&
```

```
          !( (i == g[j]->start_char ) &&
             (last_match_position[i] == g[j]->stop_char ) ) )
      {
        is_part_of_former_group = 1;
      }
    }

    if ( is_part_of_former_group &&
         ((last_match_position[i]-i)+1 >= minimum_length) )
    {
      g[extended_group_count] = (GROUP *)(malloc(sizeof(GROUP)));
      init_group(g[extended_group_count]);
      setup_group(g[extended_group_count],longest_match_str[i],
                  i,last_match_position[i]);
      extended_group_count++;
    }
  } // end : for(i=0;i<line_length;i++)

  // create word start at position i but not longest
  for(i=0;i<group_count;i++)
  {
    // have atleast 2 chars
    if ((g[i]->stop_char - g[i]->start_char)+1 >= minimum_length)
    //<=== have atleast 2 char
    {
      tmp_str[0] = buffer[g[i]->start_char];
      for(j=g[i]->start_char+1;j<g[i]->stop_char;j++)
      {
        tmp_str[j-g[i]->start_char] = buffer[j];
        tmp_str[j-g[i]->start_char+1] = '\0';

        if ( is_in_dictionary(dictionary,
             cfg->total_word_in_dict,tmp_str) >= 0 )
        {
          g[extended_group_count] = (GROUP *)(malloc(sizeof(GROUP)));
          init_group(g[extended_group_count]);
          setup_group(g[extended_group_count],tmp_str,
                      g[i]->start_char,j);
          extended_group_count++;
        }
      }
    }
  }

  // create overlap group
  for(i=0;i<group_count;i++)
  {
    for(j=i+1;j<group_count;j++)
    {
      // if group i and j have common part create more group
      if  ( g[i]->stop_char >= g[j]->start_char )
      {
       // tree pieces separate
        // create group for g[i]-g[j]
          for(k=g[i]->start_char;k<g[j]->start_char;k++)
            tmp_str[k-g[i]->start_char] = buffer[k];
          tmp_str[k-g[i]->start_char] = '\0';

          g[extended_group_count] = (GROUP *)(malloc(sizeof(GROUP)));
          init_group(g[extended_group_count]);
          setup_group(g[extended_group_count],tmp_str,
                      g[i]->start_char,g[j]->start_char-1);

          extended_group_count++;

        // create group for g[i] intersect g[j]
          for(k=g[j]->start_char;k<=g[i]->stop_char;k++)
            tmp_str[k-g[j]->start_char] = buffer[k];
          tmp_str[k-g[j]->start_char] = '\0';

          g[extended_group_count] = (GROUP *)(malloc(sizeof(GROUP)));
          init_group(g[extended_group_count]);
          setup_group(g[extended_group_count],tmp_str,
                      g[j]->start_char,g[i]->stop_char);

          extended_group_count++;

        // create group for g[j]-g[i]
          for(k=g[i]->stop_char+1;k<=g[j]->stop_char;k++)
            tmp_str[k-(g[i]->stop_char+1)] = buffer[k];
```

```
        tmp_str[k-(g[i]->stop_char+1)] = '\0';

        g[extended_group_count] = (GROUP *)(malloc(sizeof(GROUP)));
        init_group(g[extended_group_count]);
        setup_group(g[extended_group_count],tmp_str,
                    g[i]->stop_char+1,g[j]->stop_char);

        extended_group_count++;
       // end tree pieces separate

       // two pieces seperate
        if (g[i]->stop_char > g[j]->start_char)
        {
          for(l=0;l<(g[i]->stop_char - g[j]->start_char);l++)
          {
          // create front part

            for(k=g[i]->start_char;k<=g[j]->start_char+l;k++)
              tmp_str[k-g[i]->start_char] = buffer[k];
            tmp_str[k-g[i]->start_char] = '\0';

            g[extended_group_count] = (GROUP *)(malloc(sizeof(GROUP)));
            init_group(g[extended_group_count]);
            setup_group(g[extended_group_count],tmp_str,
                        g[i]->start_char,g[j]->start_char+l);
            extended_group_count++;


          // create rear part

            for(k=(g[j]->start_char+l+1);k<=g[j]->stop_char;k++)
              tmp_str[k-(g[j]->start_char+l+1)] = buffer[k];
            tmp_str[k-(g[j]->start_char+l+1)] = '\0';

            g[extended_group_count] = (GROUP *)(malloc(sizeof(GROUP)));
            init_group(g[extended_group_count]);
            setup_group(g[extended_group_count],tmp_str,
                        g[j]->start_char+l+1,g[j]->stop_char);
            extended_group_count++;

          } // end : for(l=0;l<(g[i].stop_char - g[j].start_char);l++)
        }//endif (g[i].stop_char > g[j].start_char)


       // end two pieces seperate
    } // end if  ( g[i].stop_char >= g[j].start_char )
  } // end for(j=i+1;j<group_count;j++)
} // end for(i=0;i<group_count;i++)

// remove redundant extended group

for(i=group_count;i<extended_group_count;i++)
{
  for(j=i+1;j<extended_group_count;j++)
  {
    if ( (g[i]->start_char == g[j]->start_char) &&
         (g[i]->stop_char == g[j]->stop_char) )
    {
      // move last entry to j(the dupplicate)
      group_copy(g[j],g[extended_group_count-1]);
      j--; // need this to not make checking new record move here
      extended_group_count--;
    }
  }
}

// remove stand alone group
//(group not have incomeing and outgoing edge when created graph)
for(i=group_count;i<extended_group_count;i++)
{
  have_incoming_edge = 0;
  have_outgoing_edge = 0;
  for(j=0;j<extended_group_count;j++)
  {
    // check for incoming edge
    if ( (g[j]->stop_char+1) == g[i]->start_char )
    {
      have_incoming_edge = 1;
    }
    // check for outgoing edge
```

```
            if ( (g[i]->stop_char+1) == g[j]->start_char )
            {
              have_outgoing_edge = 1;
            }
          } // end : for(j=0;j<extended_group_count;j++)

          if ( (!(have_incoming_edge) && (g[i]->start_char != 0)) ||
               (!(have_outgoing_edge) && (g[i]->stop_char != (line_length-1)) ) )
          {
            // remove a group
              // move last entry to j(the dupplicate)
              group_copy(g[i],g[extended_group_count-1]);
              i--; // need this to not make checking new record move here
              extended_group_count--;
          }
        }

        // create real group here

        for(i=0;i<extended_group_count;i++)
        {
          tmp_set = (SET *)(malloc(sizeof(SET)));
          init_set(tmp_set);
          tmp_set->head->word_number = get_num_trie(cfg,dictionary,
                                                    trie,g[i]->str,0);

          make_group(cfg,g[i],tmp_set,
                     g[i]->str,g[i]->start_char,g[i]->stop_char,
                     dictionary);
        }

  *g_c = group_count;
  *e_g_c = extended_group_count;
}

/*-----  function using in indexing process ------*/

void f_print_sep_string(FILE *fp,CONFIG_VAR *cfg,GRAPH *gph,int path[],int
path_length,int length)
{
  int i;
  char buffer[5];
  int CC = cfg->CC;//400
  int AA = cfg->AA;//1
  int BB = cfg->BB;//1
  int cut_param;
  int l,tmp;
  int flag = 0;


  fprintf(fp,"\n");
  for(i=1;i<path_length-1;i++)
  {
    l = strlen(gph->vertice[path[i]]->str);
    tmp = (int)((1.0*CC)/((l+AA)*(l+AA))) + BB;

    cut_param = (cfg->beta_factor*tmp)/l;
    if (gph->vertice[path[i]]->group_weight > cut_param)
    {
      sprintf(buffer,"*");
      flag = 1;
    }
    else
    {
      sprintf(buffer," ");
    }
    fprintf(fp,"  %s  [%s].",gph->vertice[path[i]]->str,buffer);
  }
  if (flag)
  {
    sprintf(buffer,"***");
  }
  else
  {
    sprintf(buffer,"   ");
  }

  fprintf(fp,"  : %d[%s]",length,buffer );
}
```

```
void expand2(FILE *fp,INDEX *ind,CONFIG_VAR *cfg,MY_WORD *dictionary,
             GRAPH *gph,int path[],int path_length,int length,
             TRIE_REC *trie,
             int front,int rear,CANDIDATE_LIST *c_list,int type)
{
  CANDIDATE_LIST *can_list;
  CANDIDATE_NODE *current;
  char buffer[MAXWORDLENGTH];
  char str[MAXWORDLENGTH];
  int new_front,new_rear;
  int miss,num;
  int completeness;
  unsigned int minimum_length = cfg->min_fixed_accept_length;
  int i;
  INDEX_NODE *tmp_index_node;
  INDEX_LIST *additional_index;

  additional_index = &(ind->additional_index);

  can_list = c_list;


  current = c_list->head;
  while(current->next != NULL)
  {

    strcpy(buffer,current->next->str);

    // try to group backward
    new_front = front-1;
    new_rear = rear+1;

    miss = 0;
    while ( (new_front>=1)&& (!miss) )
    {
      sprintf(str,"%s%s",gph->vertice[path[new_front]]->str,buffer);
      num = get_num_trie(cfg,dictionary,trie,str,0);
      if (num > 0)
      {
        strcpy(buffer,str);
        new_front--;
      }
      else
      {
        miss = 1;
        new_front++;
      }
    }

    miss = 0;
    // try to group forward
    while ( (new_rear<path_length-1)&& (!miss) )
    {
      sprintf(str,"%s%s",buffer,gph->vertice[path[new_rear]]->str);
      num = get_num_trie(cfg,dictionary,trie,str,0);
      if (num > 0)
      {
        strcpy(buffer,str);
        new_rear++;
      }
      else
      {
        miss = 1;
        new_rear--;
      }
    }

    // must check boundary of new_front and new rear
    if (new_front <= 0)
    {
      new_front = 1;
    }
    if (new_rear >= path_length-1)
    {
      new_rear = path_length-2;
    }

    completeness = complete_word(cfg,dictionary,trie,buffer,0);
    if (strlen(buffer) >= minimum_length)
```

```
    {
      if (completeness == 1)
      {
        fprintf(fp,"\n\t %s",buffer);
        switch(type)
        {
          case INSERTION_ERROR      : fprintf(fp,"{insertion}");
                                      break;
          case DELETETION_ERROR     : fprintf(fp,"{deletetion}");
                                      break;
          case SUBSTITUTETION_ERROR : fprintf(fp,"{substitutetion}");
                                      break;
          case TRANSPOSITION_ERROR  : fprintf(fp,"{transposition}");
                                      break;
          default                   :
                                      break;
        }
        fprintf(fp,"\t<<< ");
        for(i=new_front;i<=new_rear;i++)
        {
          fprintf(fp,"[ %s ]",gph->vertice[path[i]]->str);
        }
        fprintf(fp," >>>");
        create_index_node(buffer,
                          gph->vertice[path[new_front]]->start_char,
                          gph->vertice[path[new_rear]]->stop_char,
                          type,&tmp_index_node);
        insert_index_list(additional_index,tmp_index_node);

      }

    }


    current = current->next;
  }

}

void expand(FILE *fp,INDEX *ind,CONFIG_VAR *cfg,MY_WORD *dictionary,
            GRAPH *gph,int path[],int path_length,int length,
            TRIE_REC *trie,
            int front,int rear,CANDIDATE *can)
{
  expand2(fp,ind,cfg,dictionary,
          gph,path,path_length,length,
          trie,
          front,rear,&(can->insertion),INSERTION_ERROR);
  expand2(fp,ind,cfg,dictionary,
          gph,path,path_length,length,
          trie,
          front,rear,&(can->deletetion),DELETETION_ERROR);
  expand2(fp,ind,cfg,dictionary,
          gph,path,path_length,length,
          trie,
          front,rear,&(can->substitutetion),SUBSTITUTETION_ERROR);
  expand2(fp,ind,cfg,dictionary,
          gph,path,path_length,length,
          trie,
          front,rear,&(can->transposition),TRANSPOSITION_ERROR);

}

void find_more(FILE *fp,INDEX *ind,CONFIG_VAR *cfg,MY_WORD *dictionary,
               GRAPH *gph,int path[],int path_length,int length,
               TRIE_REC *trie)
{
  int i;
  int CC = cfg->CC;//400
  int AA = cfg->AA;//1
  int BB = cfg->BB;//1
  int cut_param;
  int str_len,tmp;
  int flag = 0;
  CANDIDATE can;
  char buffer[MAXWORDLENGTH];
  char str[MAXWORDLENGTH];
  int num;
  int forward;
  int backward;
```

```
int miss;


fprintf(fp,"\n additional words \n");
for(i=1;i<path_length-1;i++)
{
  str_len = strlen(gph->vertice[path[i]]->str);
  tmp = (int)((1.0*CC)/((str_len+AA)*(str_len+AA))) + BB;

  cut_param = (cfg->beta_factor*tmp)/str_len;
  if (gph->vertice[path[i]]->group_weight > cut_param)
  {
    //  try to merge with adjacent node

      // try to merge forward
      sprintf(buffer,"%s",gph->vertice[path[i]]->str);
      forward = i+1;
      backward = i;
      miss = 0;
      while ( (forward<path_length-1) && (!miss) )
      {
        sprintf(str,"%s%s",buffer,gph->vertice[path[forward]]->str);
        num = get_num_trie(cfg,dictionary,trie,str,0);
        if (num > 0)
        {
          // need to check at this point aslo coz it miss in some case
          // but it slow down alot
          strcpy(buffer,str);
          init_candidate(&can);
          create_candidate(cfg,dictionary,trie,buffer,&can);
          expand(fp,ind,cfg,dictionary,
                 gph,path,path_length,length,
                 trie,
                 i,forward,&can);
          //f_print_candidate_list(fp,&(can.insertion));
          free_candidate(&can);

          forward++;
        }
        else
        {
          init_candidate(&can);
          create_candidate(cfg,dictionary,trie,str,&can);
          expand(fp,ind,cfg,dictionary,
                 gph,path,path_length,length,
                 trie,
                 i,forward,&can);
          //f_print_candidate_list(fp,&(can.insertion));
          free_candidate(&can);
          forward--;
          miss = 1; // force exit loop
        }
      }

      // try to merge backward after dead
      backward = i-1;
      miss = 0;
      while ( (backward>=1) && (!miss) )
      {
        sprintf(str,"%s%s",gph->vertice[path[backward]]->str,buffer);
        num = get_num_trie(cfg,dictionary,trie,str,0);
        if (num > 0)
        {
          // need to check at this point aslo coz it miss in some case
          // but it slow down alot
          strcpy(buffer,str);
          init_candidate(&can);
          create_candidate(cfg,dictionary,trie,buffer,&can);
          expand(fp,ind,cfg,dictionary,
                 gph,path,path_length,length,
                 trie,
                 backward,forward,&can);
          //f_print_candidate_list(fp,&(can.insertion));
          free_candidate(&can);

          backward--;
        }
        else
        {
```

```
            init_candidate(&can);
            create_candidate(cfg,dictionary,trie,str,&can);
            expand(fp,ind,cfg,dictionary,
                   gph,path,path_length,length,
                   trie,
                   backward,forward,&can);
            //f_print_candidate_list(fp,&(can.insertion));
            free_candidate(&can);
            backward++;
            miss = 1; // force exit loop
          }
        }

//
        // try to merge backward
        sprintf(buffer,"%s",gph->vertice[path[i]]->str);
        forward = i;
        backward = i-1;
        miss = 0;
        while ( (backward>=1) && (!miss) )
        {
          sprintf(str,"%s%s",gph->vertice[path[backward]]->str,buffer);
          num = get_num_trie(cfg,dictionary,trie,str,0);
          if (num > 0)
          {
            // need to check at this point aslo coz it miss in some case
            // but it slow down alot
            strcpy(buffer,str);
            init_candidate(&can);
            create_candidate(cfg,dictionary,trie,buffer,&can);
            expand(fp,ind,cfg,dictionary,
                   gph,path,path_length,length,
                   trie,
                   backward,i,&can);
            //f_print_candidate_list(fp,&(can.insertion));
            free_candidate(&can);

            backward--;
          }
          else
          {
            init_candidate(&can);
            create_candidate(cfg,dictionary,trie,str,&can);
            expand(fp,ind,cfg,dictionary,
                   gph,path,path_length,length,
                   trie,
                   backward,i,&can);
            //f_print_candidate_list(fp,&(can.insertion));
            free_candidate(&can);
            backward++;
            miss = 1; // force exit loop
          }
        }

        // try to merge forward after dead
        forward = i+1;
        miss = 0;
        while ( (forward<path_length-1) && (!miss) )
        {
          sprintf(str,"%s%s",buffer,gph->vertice[path[forward]]->str);
          num = get_num_trie(cfg,dictionary,trie,str,0);
          if (num > 0)
          {
            // need to check at this point aslo coz it miss in some case
            // but it slow down alot
            strcpy(buffer,str);
            init_candidate(&can);
            create_candidate(cfg,dictionary,trie,buffer,&can);
            expand(fp,ind,cfg,dictionary,
                   gph,path,path_length,length,
                   trie,
                   backward,forward,&can);
            //f_print_candidate_list(fp,&(can.insertion));
            free_candidate(&can);

            forward++;
          }
          else
          {
            init_candidate(&can);
```

```
            create_candidate(cfg,dictionary,trie,str,&can);
            expand(fp,ind,cfg,dictionary,
                   gph,path,path_length,length,
                   trie,
                   backward,forward,&can);
            //f_print_candidate_list(fp,&(can.insertion));
            free_candidate(&can);
            forward--;
            miss = 1; // force exit loop
          }
        }

    }// end if :: if (gph->vertice[path[i]]->group_weight > cut_param)
  }
}

void process(CONFIG_VAR *cfg,MY_WORD *dictionary,void *ngram_table,
             void *memspace,TRIE_REC *trie)
{
  FILE *input_fp,*output_fp;
  FILE *graph_fp,*debug_fp;
  FILE *index_fp;

  int i;

  int  group_count;
  int  extended_group_count;
  char buffer[MAXLINELENGTH];
  GROUP *g[MAXGROUPINLINE];

  GRAPH *std_graph;// standard graph (unfix graph use as control)
  int std_path[MAXPATHLENGTH];
  int std_path_length;// number of edge in path
  int std_length;
  INDEX ind;
  int mode = 1; // if 1 will remove the word same as the word in base index list
  // from additional index list

  std_graph = (GRAPH *)(malloc(sizeof(GRAPH)));


  // open input file
  if( (input_fp  = fopen( "unsep.txt", "rt" ))  == NULL )
    printf( "\nThe 'input.txt' file was not opened\n" );
  else
  {
    printf( "\nThe 'input.txt' file was opened\n" );
    if ( (output_fp  = fopen( "output.txt", "wt+" )) == NULL ||
         (debug_fp  = fopen( "debug.txt", "wt+" )) == NULL ||
         (graph_fp  = fopen( "graph.txt", "wt+" )) == NULL ||
         (index_fp  = fopen( "index.txt", "wt+" )) == NULL )
      printf( "\nSome of output files were not opened\n" );
    else
    {
      printf( "\nAll output files were opened\n" );

      while(!feof(input_fp))
      {
        fscanf(input_fp,"%s\n",buffer);
        //printf("\n line  : %s",buffer);

        // separate sentence and prepare information
        init_index(&ind);
        sep(cfg,dictionary,ngram_table,memspace,trie,
            g,&group_count,&extended_group_count,buffer);

        // create graph
        //printf("\n creating graph");
        init_graph(std_graph,extended_group_count);
        crate_graph(std_graph,g,extended_group_count);
        f_print_graph(graph_fp,std_graph);

        // shortest path
        //printf("\n shortest path");
        shortest_path(std_graph,&std_length,&std_path_length,std_path);
        f_print_shortest_path(output_fp,std_graph,&std_length,
                              &std_path_length,std_path);
        // detect and fixing error
        //fprintf(debug_fp,"\n\n Processing error detection for line");
        //fprintf(debug_fp,"\n %s",buffer);
```

```
        // indexing

        fprintf(index_fp,"\n sentence : %s",buffer);
        f_print_sep_string(index_fp,cfg,std_graph,std_path,std_path_length,std_length);
        fprintf(index_fp,"\n base index");
        create_base_index(index_fp,&(ind.base_index),cfg,dictionary,trie,
                          g,group_count,extended_group_count,
                          std_graph,std_path,std_path_length,std_length);

        //fixing_error(debug_fp,cfg,dictionary,ngram_table,memspace,
        //             std_graph,std_path,std_path_length,std_length);
        f_print_sep_string(debug_fp,cfg,std_graph,std_path,std_path_length,std_length);

        find_more(debug_fp,&ind,cfg,dictionary,std_graph,std_path,std_path_length,
                  std_length,trie);
        fprintf(debug_fp,"\n---------------------------------------");
        if (mode == 1)
        {
          remove_duplicate(&ind);
        }
        fprintf(index_fp,"\n additional index");
        f_print_index_list(index_fp,&(ind.additional_index));
        fprintf(index_fp,"\n-----------------------------------------------\n");

        // free memory
        free_index(&ind);
        for(i=0;i<extended_group_count;i++)
        {
          free_group(g[i]);
          free(g[i]);
        }
        printf("#");
      }

      fclose(output_fp);
      fclose(graph_fp);
      fclose(debug_fp);
      fclose(index_fp);

    }

    fclose(input_fp);
  }

  free(std_graph);
}

/*--------  main program ----------*/
void main()
{
  MY_WORD *dictionary;
  TRIE_REC *trie;
  void *ngram_table,*memspace;
  char str[MAXWORDLENGTH];
  CANDIDATE can;
  CONFIG_VAR cfg;

  // initialize dictionary
  init_dictionary(&cfg,&dictionary);
  // read config file
  open_config_file(&cfg);
  print_config(&cfg);

  //  n-gram part
  // allocate memory for n-gram table(ok)
  alloc_memoryspace_for_ngram_table(&cfg,&ngram_table);
  memspace = (void *)(malloc(sizeof(NODE)*(250000)));
  // read n-gram table
  read_ngram_table(&cfg,ngram_table,memspace);

// trie part
  make_trie(&cfg,dictionary,&trie);
  printf("\n Processing");
  process(&cfg,dictionary,ngram_table,memspace,trie);

  printf("\n Processing completed");
  printf("\n Freeing trie : please wait (this may take a while about 50 secs :p)");
  free_trie(trie);
  free(trie);
```

```
  free(dictionary);

  free(ngram_table);
  free(memspace);

  printf("\n Total number of nodes used : %d\n",number_of_nodes_used);
  printf("\n Total number of candidate nodes used : %d\n",
            number_of_candidate_nodes_used);
  printf("\n Total number of index nodes used : %d\n",number_of_index_nodes_used);
  printf("\n Total number of trie leaf nodes used : %d\n",
            trie_leaf_node_counter);
  printf("\n Total number of trie internal nodes used : %d\n",
            trie_internal_node_counter);
  printf("\n Total number of trie leaf list used : %d\n",trie_leaf_list_counter);

}
```



สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

## ภาคผนวก ค.

**ตัวอย่างข้อความที่นำมาทดสอบ**

การประชมทางวิชาการโครงการวิจัยและพัฒนาปีงบประมาณ
ศูนย์เทคโนโซยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ
ศูนย์เทคโนโยลีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ
กระทรงวงวิทยาศาสตร์เทคโนโลยีและการพลังงาน
กระทรงวิทยาศาสตร์เทคโนโลยีและการพลังงาน
รัฐนมตรีว่าการกระทรวงวิทยาศาสตร์
ปรเทศไทยได้มีการปรับเปลี่ยนโครงสร้างในการพัฒนาเศรษฐกิจของประเทศ
จากประเทศเกตรกรรมไปสู่ความเป็นประเทศอุตสาหกรรม
ในการดำเนินการเพื่อให้บรรลุวัตถุปาระสงค์
ในการดำเนินการเพื่อให้บรรลุวัตถุปะสงค์
จะต้องอาศัยปัจจัยพื้นฐานหลาวประการในการเป็นตัวเร่ง
การพัฒนาเทคโนโลยีที่ใช้ในการผลิของภาคอุตสาหกรรม
มนุษย์ส่วนใหญ่ในวังคมจะต้องประสบกับปัญหาสังคม
ไม่แกระทบกระเทือน
ไม่ระทบกระเทือน
เพื่อให้ปัญหานั้นบรรทาเบาบางลง
การจราจรคิดขัด
ความจริงไม่ได้เป็นหลักประกนว่าจะแก้ปัญหาได้
ความล้มเหลวของกฎเกณฑ์ที่สับคมกำหนด
ความล้มเหลวของกฎเกณฑ์ที่สังคมกำหด
แต่เป็นเพียงสภาวการณ์ที่ไม่สอดล้องกับคุณค่าที่กลุ่มยึดถือ
แต่เป็นเพียงสภาวการณ์ที่ไม่สอดคลองกับคุณค่าที่กลุ่มยึดถือ
ในระบบปรมวลกฎหมาย
การฆ่าตัวตายในทัศนะของนักสับคมวิทยา
ความเสป่าเปลี่ยว
การเปลียนแปลงสภาพแวดล้อม
การเปลี่ยนแปลงสภาแพวดล้อม

งานวิจัยเพื่อเพิ่มประสิทธิภาพของวิธีการเรียนรู้โดยกำหนดการเชิงพันธุกรม
งานวิจัยเพื่อเพิ่มประสิทธิภาของวิธีการเรียนรู้โดยกำหนดการเชิงพันธุกรรม
งานยิจัยเพื่อเพิ่มประสิทธิภาพของวิธีการเรียนรู้โดยกำหนดการเชิงพันธุกรรม
งานวิจัยเพื่อเพิ่มประสิทธิภาพองวิธีการเรียนรู้โดยกำหนดการเชิงพันธุกรรม
งานวิจัยเพื่อเพิ่มประสิทธิภพของวิธีการเรียนรู้โดยกำหนดการเชิงพันธุกรรม
งานวิจัยเพื่อเพิ่มประสิทธิภาพของวิธีกรเรียนรู้โดยกำหนดการเชิงพันธุกรรม
งานวิจัยเพื่อเพิ่มปีระสิทธิภาพของวิธีการเรียนรู้โดยกำหนดการเชิงพันธุกรรม
งานวิจัยเพื่อเพิ่มปะสิทธิภาพของวิธีการเรียนรู้โดยกำหนดการเชิงพันธุกรรม
บานวิจัยเพื่อเพิ่มประสิทธิภาพของวิธีการเรียนรู้โดยกำหนดการเชิงพันธุกรรม
งานวิจัยเพื่อเพิ่มประสิทธิภาพของวิธีการเรียนรู้โดยกำหนดการเชอิงพันธุกรรม
งานวิจัยเพื่อเพิ่มประสิทธิภาพของวิธีการเรียนรู้โดยกำหนการเชิงพันธุกรรม
งานวิจัยเพื่อเพิ่มประสิทธิภาพของวิธีการเรียนรู้โดยกำหนอการเชิงพันธุกรรม
งานวิจัยเพื่อเพิ่มประสิทธิภาพของวิธีการเรียนรู้โดยกำหนดหารเชิงพันธุกรรม
แนวทงในการเพิ่มประสิทธิภาพ
แนวางในการเพิ่มประสิทธิภาพ
แนวาทงในการเพิ่มประสิทธิภาพ
นวทางในการเพิ่มประสิทธิภาพ
จากลการทดลองพบว่า
จากลผการทดลองพบว่า
จากผกลารทดลองพบว่า
จากผลการทอลองพบว่า
ประสบการณ์ที่หุ่นยนต์ได้เนียนรู้
ประสบการณ์ที่หุนยนต์ได้เรียนรู้
แประสบการณ์ที่หุ่นยนต์ได้เรียนรู้
ระสบการณ์ที่หุ่นยนต์ได้เรียนรู้
แระสบการณ์ที่หุ่นยนต์ได้เรียนรู้
เทคนิคบางดย่างทางด้านปัญญาประดิษฐ์
เทคนิคบาวงอย่างทางด้านปัญญาประดิษฐ์
กระบวนการวิวัฒน่าการทางธรรมชาติ
กระบวนการวิวัฒนการทางธรรมชาติ
กระบวนการวิวัฒนมการทางธรรมชาติ

กระขบวนการวิวัฒนาการทางธรรมชาติ

กระวนการวิวัฒนาการทางธรรมชาติ

ค้นหาคำตอบของปัญห่าโดยจำลองแบบมาจาก

ค้นหาคำอบของปัญหาโดยจำลองแบบมาจาก

ค้นหาคำจอบของปัญหาโดยจำลองแบบมาจาก

ค้นหาคำตบของปัญหาโดยจำลองแบบมาจาก

การสร้างกลุ่มประชากรของผบเฉลยอย่างสุ่ม

การสร้างกบลุ่มประชากรของผลเฉลยอย่างสุ่ม

ได้รับแรงบันดาซลใจจากงานวิจัย

ได้รับแรงบันดาบใจจากงานวิจัย

ได้รับแรงบันดาใลจจากงานวิจัย

ได้รับแรงบนดาลใจจากงานวิจัย

การประเมินค่าความเหมาะของผลเฉย

การระเมินค่าความเหมาะของผลเฉลย

สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

# ประวัติผู้วิจัย

นายวรวัฒน์ วรศิลป์ เกิดวันที่ 3 พฤษภาคม 2516 ที่อำเภอเมือง จังหวัดมหาสารคาม สำเร็จการศึกษาปริญญาตรีวิทยาศาสตร์บัณฑิต สาขา วิทยาการคอมพิวเตอร์ จาก ภาควิชาคณิตศาสตร์ คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2538 และเข้าศึกษาต่อในหลักสูตรวิทยาศาสตร์มหาบัณฑิตที่จุฬาลงกรณ์มหาวิทยาลัย เมื่อ พ.ศ. 2539

จุฬาลงกรณ์มหาวิทยาลัย