โครงสร้างข้อมูลพจนานุกรมเพื่อการตัดคำและอัลกอริธึมที่ใช้ในการหาคำอ่านไทย

Data structure dictionary for Thai word segmentation and Algorithm for convert to phonetic symbols

โดย

นายนันทชัย สมัญญาภรณ์ รหัสประจำตัว 41014216

นายสรวุฒิ กอสุวรรณศิริ รหัสประจำตัว 41014446

อาจารย์ที่ปรึกษา

รศ.ดร.บุญธีร์ เครือตราชู

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาวิศวกรรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

ปริญญานิพนธ์ ปีการศึกษา 2545

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง โครงสร้างข้อมูลพจนานุกรมเพื่อการตัดคำและอัลกอริธึมที่ใช้ในการหาคำอ่านไทย

Data structure dictionary for Thai word segmentation and Algorithm for convert to phonetic symbols

ผู้จัดทำ

นายนันทชัย สมัญญาภรณ์ รหัสประจำตัว 41014216

นายสรวุฒิ กอสุวรรณศิริ รหัสประจำตัว 41014446

อาจารย์ที่ปรึกษา

(รศ.ดร.บุญธีร์ เครือตราชู)

โครงสร้างข้อมูลพจนานุกรมเพื่อการตัดคำและอัลกอริธึมที่ใช้ในการหาคำอ่านไทย

นายนันทชัย สมัญญาภรณ์
นายสรวุฒิ กอสุวรรณศิริ
รศ.ดร.บุญธีร์ เครือตราชู อาจารย์ที่ปรึกษา

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้เป็นการศึกษาเพื่อหาโครงสร้างข้อมูลที่ดีเหมาะสม เพื่อใช้ในการจัดเก็บพจนานุกรมภาษาไทย เพื่อใช้ในการตัดคำออกจากประโยคและยังรวมการหาคำผิด โดยคำนึงถึงคุณสมบัติต่างๆ คือ เป็นโครงสร้างข้อมูลที่มีความรวดเร็วในการค้นคำ มีความสามารถในการรับอินพุตที่มีความยาวไม่แน่นอน มีการใช้พื้นที่หน่วยความจำได้อย่างมีประสิทธิภาพ และที่สำคัญคือสามารถนำไปประยุกต์ใช้งานได้ง่าย ซึ่งโครงสร้างข้อมูลที่ได้นำมาศึกษานั้นได้แก่ hash function , 2-3 tree , access trie , trie ternary tree และ burst trie และในในปริญญานิพนธ์นี้ยังเกี่ยวกับการหาอัลกอริธึมที่ใช้ในการหาคำอ่าน โดยจะถูกแบ่งออกเป็น 2 ขั้นตอนหลักๆ คือ การแยกพยางค์ออกจากคำ และ การแปลงพยางค์ให้กลายเป็นสัญลักษณ์การออกเสียง โดยในขั้นตอนการแยกพยางค์นั้นเราจำเป็นจะต้องรู้หลักภาษาไทยเพื่อใช้ในการแยกพยางค์ และในการแปลงพยางค์ให้เป็นสัญลักษณ์การออกเสียงนั้นก็ต้องรู้หลักภาษาเพื่อใช้ในการแยกว่าตัวใดเป็นพยัญชนะต้น สระ ตัวสะกด และวรรณยุกต์

Data structure dictionary for Thai word segmentation

and Algorithm for convert to phonetic symbols

Mr. Nantachai Samanyaporn

Mr. Sorawut Korsuwansiri

Assoc.Prof.Dr.Boontee Kruatrachue Advisor

ABSTRACT

This thesis is the researching to find data structure that suitable for Thai dictionary to use in Thai word segmentation and include find mistake word. These data structures should have these properties : find word fast , receive input any length , have efficiency to use memory , and ease of use. These data structures that considered is hash function , 2-3 tree , access trie , trie , ternary tree , and burst tree. And in this thesis is researching algorithm to extract syllable from Thai word and convert them to phonetic symbols. In process of extracting syllable from Thai word we must know Thai language rules. And in process converting to phonetic symbols we must know Thai language rules to find initial letter , vowel letter , final letter , and level of sound.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้คงไม่อาจสำเร็จได้ด้วยดี หากไม่ได้รับความช่วยเหลือและกำลังใจจากหลายๆ ท่านด้วยกัน บุคคลแรกที่ต้องกล่าวถึงคือ รศ.ดร.บุญธีร์ เครือตราชู อาจารย์ที่ปรึกษาวิทยานิพนธ์ เพราะเป็นท่านที่สำคัญที่ทำให้สามารถเริ่มทำโครงงานชิ้นนี้ได้และให้คำแนะนำต่างๆ เพื่อนๆ 4D ทุกคนที่ได้คุยได้เล่นกัน

และขอขอบพระคุณสำหรับบุคคลสำคัญที่สุดของข้าพเจ้าที่ทำให้มีวันนี้ ก็คือ บิดา มารดา ผู้ให้กำเนิดและเลี้ยงดูข้าพเจ้ามาตั้งแต่ยังเล็ก อีกทั้ง พี่น้อง และ ญาติๆ ที่เป็นกำลังใจให้และอีกหลายๆท่านที่ไม่ได้กล่าวถึง ณ ที่นี้ ข้าพเจ้าขอระลึกในพระคุณของท่านและกราบขอบพระคุณมา ณ ที่นี้

นันทชัย สมัญญาภรณ์

สรวุฒิ กอสุวรรณศิริ

สารบัญ

สารบัญตาราง

สารบัญภาพ

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

โครงงานนี้เกิดจากความต้องการโปรแกรมที่สามารถตัดคำไทย และสามารถค้นหาคำที่ผิดได้ และหาโครงสร้างข้อมูลที่เหมาะสมเพื่อใช้ในการทำพจนานุกรมเพื่อการตัดคำและหาคำผิด และยังสามารถที่จะหาคำอ่านพร้อมทั้งแปลงให้อยู่ในรูปสัญลักษณ์สากล ที่คนชาติอื่นๆ เห็นแล้วสามารถอ่านออกเสียงได้ใกล้เคียงกับคำอ่านจริงได้

1.2 วัตถุประสงค์ของโครงงาน

1.2.1 ศึกษาเรื่องโครงสร้างข้อมูลแบบต่างๆ ได้แก่ Hash Function, 2-3 Tree, Access Trie, Trie, Ternary Tree, Burst Tree

1.2.2 ศึกษาเรื่องอัลกอริธึมที่ใช้ในการตัดคำไทย และค้นหาคำผิด

1.2.3 ศึกษาถึงหลักภาษาไทยเพื่อใช้ในการหาคำอ่าน

1.2.4 ศึกษาถึงสัญลักษณ์ phonetic เพื่อใช้ในการแปลงคำอ่านมาเป็นสัญลักษณ์สากล

1.3 ขอบเขตของโครงงาน

เขียนโปรแกรมเพื่อใช้โครงสร้างข้อมูลแบบต่างๆ สำหรับการทำพจนานุกรม เพื่อทดสอบหาข้อดีข้อเสียของโครงสร้างข้อมูลแต่ละแบบได้ พร้อมทั้งเขียนโปรแกรมเพื่อใช้ในการหาคำอ่านไทย(แยกพยางค์) และโปรแกรมเพื่อใช้ในการแปลงคำอ่านไทยเป็นสัญลักษณ์การออกเสียงได้

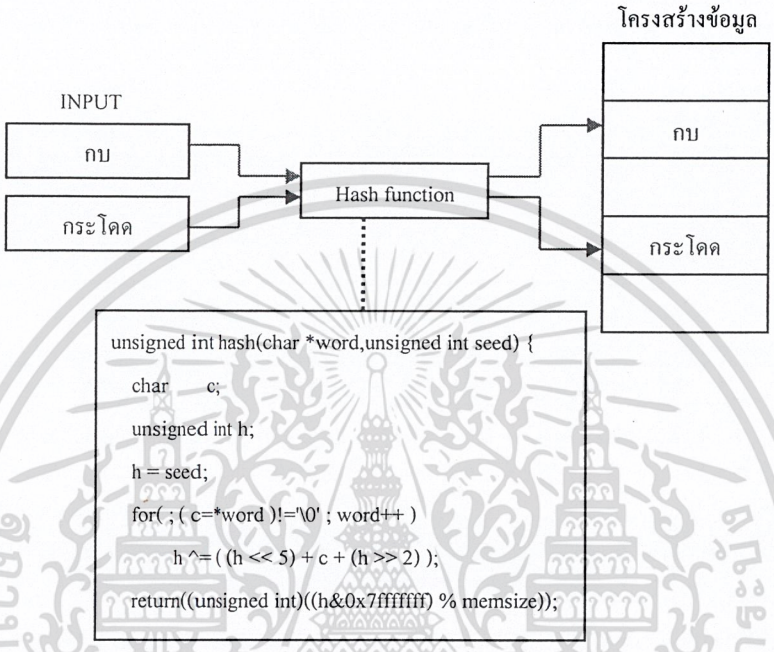
1.4 วิธีการดำเนินการ

เริ่มจากการศึกษาและค้นคว้าอัลกอริธึม, โครงสร้างข้อมูล และข้อมูลที่เกี่ยวข้องกับโครงสร้างภาษาไทยต่างๆ เพื่อใช้ในการเขียนฟังก์ชั่นต่างๆ และพัฒนาโปรแกรม โดยมีการอธิบายรายละเอียดการทำงานของแต่ละฟังก์ชั่น, การวัดผลการทดลองการใช้อัลกอริธึมและโครงสร้างข้อมูลแต่ละแบบว่ามีความเร็วในการทำงานมากน้อยเพียงใด และสรุปการการทดลองต่างๆ รวมทั้งข้อดีและข้อเสียของแต่ละแบบ

บทที่ 2
ทฤษฎีโครงสร้างข้อมูลแบบต่าง ๆ

2.1 โครงสร้างข้อมูลแบบ Hash

โครงสร้างข้อมูลแบบ Hash มีลักษณะเป็นอาร์เรย์ 1 มิติที่ประกาศจำนวน record ไว้มากกว่าจำนวนของข้อมูล และใช้สมการที่มีเซตของข้อมูลเป็นโดเมนในการค้นหาตำแหน่งของข้อมูล



รูปที่ 2-1 แสดงลักษณะโครงสร้างข้อมูลแบบ hash

2.1.1 วิธีการค้นหาข้อมูล

การค้นหาข้อมูลในโครงสร้างข้อมูลแบบ hash จะเอาข้อมูลมาผ่านการคำนวณในฟังก์ชั่น hash เพื่อให้ได้ตำแหน่งของข้อมูล หากตำแหน่งที่ได้มามีข้อมูลไม่ตรงกับที่ต้องการค้นหาก็จะทำการ hash ใหม่ เรียกว่า การ rehash จนกว่าจะพบข้อมูล ในกรณีที่ไม่มีข้อมูลที่ต้องการค้นหาอยู่ในโครงสร้างข้อมูล การ hash จะจบลงเมื่อพบตำแหน่งที่ไม่มีข้อมูล หรือมีจำนวนครั้งในการ rehash เกินกว่าค่าการ rehash มากที่สุด ที่จดจำไว้ในขั้นตอนการใส่ข้อมูล

2.1.2 วิธีการเก็บข้อมูล

การเก็บข้อมูลในโครงสร้างข้อมูลแบบ hash จะเอาข้อมูลมาผ่านการคำนวณในฟังก์ชั่น hash เพื่อให้ได้ตำแหน่งของข้อมูล แล้วเก็บข้อมูลลงในตำแหน่งนั้น หากในตำแหน่งนั้นมีข้อมูลอยู่แล้วก็จะเอาผลการ hash มาทำการ rehash เพื่อให้ได้ตำแหน่งใหม่ จนกว่าจะพบตำแหน่งที่สามารถเก็บข้อมูลได้

2.1.3 ฟังก์ชั่น hash ที่ใช้

hash ฟังก์ชั่นที่ใช้เป็นฟังก์ชั่น hash ในระดับบิตแบบ bitwise hash ซึ่งมีสมการดังนี้

ค่า hash = (ค่าเริ่มต้น) XOR ((ค่าเริ่มต้น x 2^5) + c + (ค่าเริ่มต้น / 2^2));

โดย c เป็นตัวอักษรแต่ละตัวในสตริง

ซึ่งเขียนเป็นโค้ดภาษา C ได้ดังนี้

```
unsigned int hash(char *word, unsigned int seed){
   char   c;
   unsigned int h;
   h = seed;
   for( ; ( c=*word )!='\0' ; word++ )
   {
         h ^= ( (h << 5) + c + (h >> 2) );
   }
   return((unsigned int)((h&0x7fffffff) % memsize));
}
```

2.1.4 ความเร็ว , ข้อดี, ข้อเสีย

ความเร็วในการจัดเก็บข้อมูล 0.07 วินาที

ความเร็วในการค้นหาคำ (แบบ Successful) 0.21 วินาที

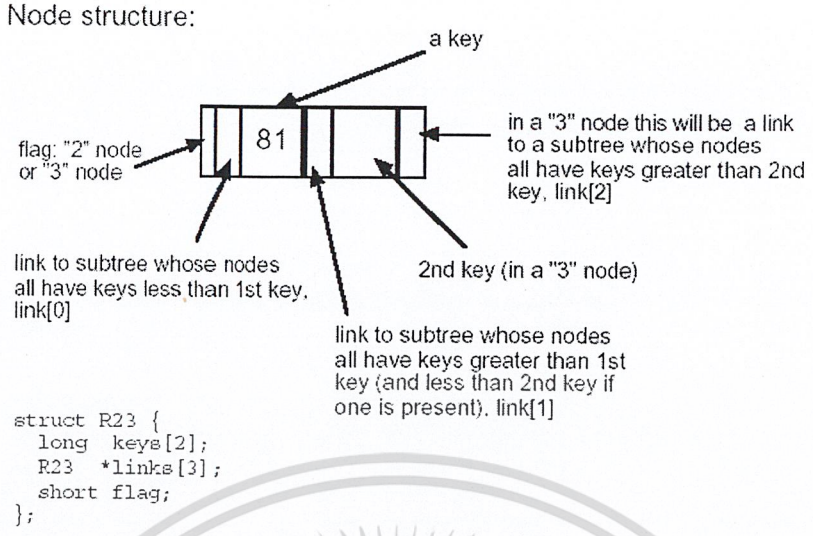
ข้อดี มีความเร็วในการทำงานสูง, ง่ายในการประยุกต์ใช้งานหรือแก้ไขเพิ่มเติม, ใช้เนื้อที่หน่วยความจำในปริมาณคงที่ และสามารถจำกัดขอบเขตการใช้หน่วยความจำได้, เหมาะกับ input ทั้งแบบ ordered list , unordered list และ reverse ordered list

ข้อเสีย ไม่เหมาะสำหรับการค้นหาแบบรับ input ทีละตัวอักษร, ใช้หน่วยความจำค่อนข้างมาก และสิ้นเปลืองไปกับ record ที่ไม่มีข้อมูลมาก, ไม่รองรับการค้นหาคำเหมือนคำใกล้เคียง, ในกรณีที่คำมีความยาวกว่าที่กำหนดไว้ จะไม่สามารถเก็บข้อมูลได้อย่างถูกต้อง

2.2 โครงสร้างข้อมูลแบบ 2-3 tree

โครงสร้างข้อมูลแบบ 2-3 tree เป็นโครงสร้างข้อมูลประเภท Tree ในแต่ละโหนดของโครงสร้างข้อมูลแบบ 2-3 tree จะมีข้อมูลอยู่ 1 หรือ 2 ตัวก็ได้ และมี branch อยู่ 2 หรือ 3 branch หาก node ใดมีข้อมูล 1 ตัว ก็จะมี 2 branch และ ที่ว่างสำหรับเก็บข้อมูลได้อีก 1 ตัว และใน node ที่ มีข้อมูล 2 ตัวก็จะมี branch อยู่ 3 branch และค่าของข้อมูลทางซ้ายจะน้อยกว่าทางขวาเสมอ

2-3 tree มีลักษณะเป็น balance tree คือในแต่ละ leaf จะมี depth เท่ากันเสมอ ทำให้ tree ทำงานได้อย่างมีประสิทธิภาพ



รูปที่ 2-2 แสดงโครงสร้างข้อมูลของ 2-3 tree

2.2.1 การค้นหาข้อมูล

วิธีการค้นหาข้อมูลใน 2-3 tree จะคล้ายกับการค้นหาข้อมูลใน binary tree ในการค้นหาสตริง s จะเริ่มต้นที่ root โดยจะนำข้อมูลที่ค้นหามาเปรียบเทียบกับข้อมูลที่เก็บอยู่ใน root ซึ่งก็คือ a และ b ซึ่งการเปรียบเทียบจะมีเงื่อนไขดังนี้

[a | b]

หาก s มีค่าตรงกับ a หรือ b ก็จะreturn ค่าที่ตรงออกมา

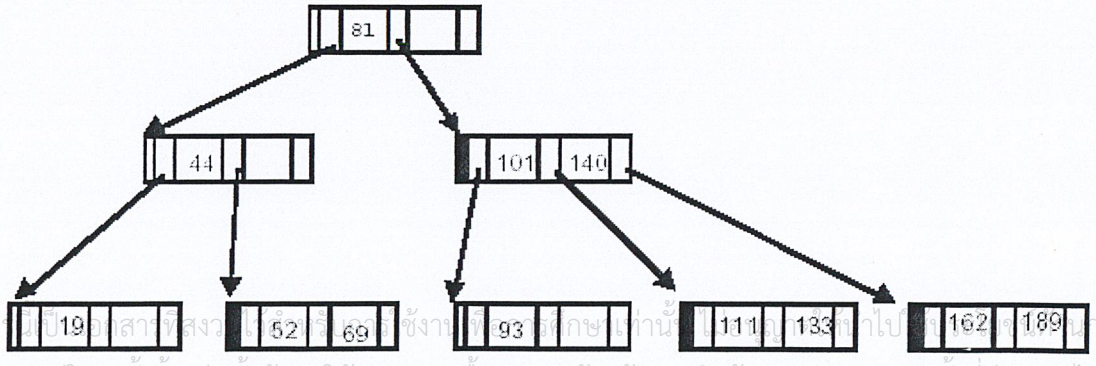
หาก s มีค่าน้อยกว่า a ก็จะลงไปค้นหาต่อใน child ทางซ้าย

หาก s มีค่ามากกว่า a แต่น้อยกว่า b หรือ b ไม่มีข้อมูล ก็จะลงไปค้นหาต่อใน child กลาง

หาก s มีค่ามากกว่า b ก็จะลงไปค้นหาต่อใน child ทางขวา

2.2.2 การเก็บข้อมูล

วิธีการเก็บข้อมูลใน 2-3 tree จะมีความซับซ้อนมากกว่า binary tree เพราะมีการปรับโครงสร้างของตัวเองให้มีความสมดุลตลอดเวลา ซึ่งจะขออธิบายแยกเป็นกรณี ดังนี้



รูปที่ 2-3 แสดงการเก็บข้อมูลของ 2-3 tree (1)

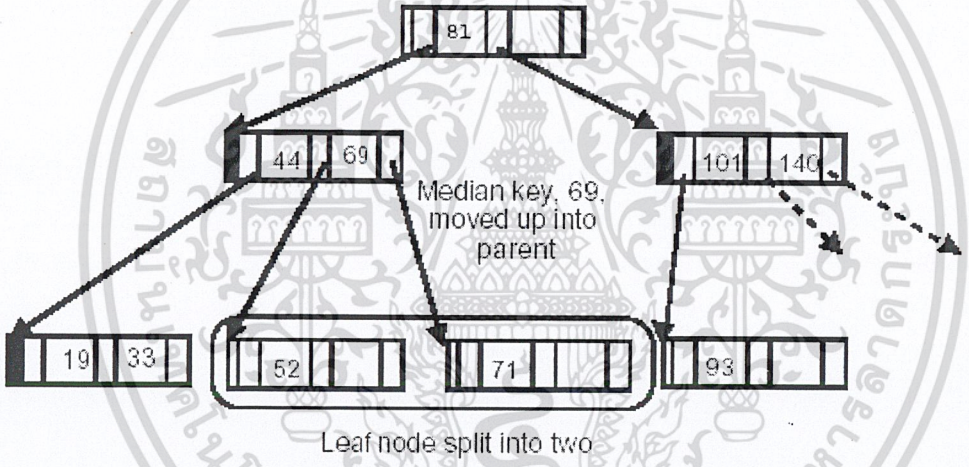
กรณีที่ 1 การ insert key 33

ในกรณีนี้จะเริ่มต้นที่ root คือ 81 และทำการค้นหาตำแหน่งด้วยวิธีเดียวกับ binary tree จนมาหยุดที่ node 19 ปรากฏว่ามีที่ว่างเหลืออยู่ จึง insert key 33 ลงในตำแหน่ง b ของ node 19

กรณีที่ 2 การ insert key 71

ในกรณีนี้จะพบปัญหาเล็กน้อยในการ insert เมื่อเราค้นหาตำแหน่งที่ควร insert key 71 ปรากฏว่าเป็น node 52 ซึ่งไม่มีที่ว่างสำหรับ key ใหม่แล้ว ในกรณีเช่นนี้ node 52 ต้องทำการ "แตกตัว" ออกเป็น 3 ส่วนคือ 52 , 69 และ 71 หลังจากนั้นจึงรวมเข้าด้วยกันใหม่โดยให้ key กลางเป็น root ของ sub tree โดยมี

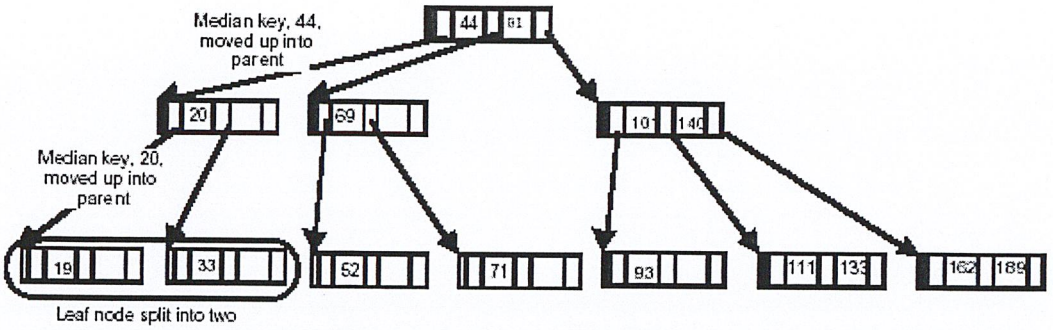
ค่าที่น้อยที่สุดเป็น child ทางซ้าย และค่าที่มากที่สุดเป็น child ทางขวา แล้วจึงนำ key กลางไปรวมกับ parent ซึ่งก็คือ node 44



รูปที่ 2-4 แสดงการจัดเก็บข้อมูลของ 2-3 tree (2)

กรณีที่ 3 การ insert key 20

ในกรณีนี้จะพบปัญหาที่ซับซ้อนมากยิ่งขึ้น เพราะ การ insert key 20 ทำให้ node [19,33] ต้องแตกตัวและเมื่อนำ key กลางไป insert ที่ โหนด [44,69] ก็ทำให้โหนดนี้ต้องแตกตัวอีกชั้นหนึ่ง ทำให้ได้ sub tree ขนาดใหญ่ขึ้นโดยมี 44 เป็น root และ 20 และ 69 เป็น child ทางซ้ายและขวา เมื่อนำ 44 ไปแทรกเข้ากับ node 81 ก็จะได้ผลลัพธ์ออกมาดังภาพ

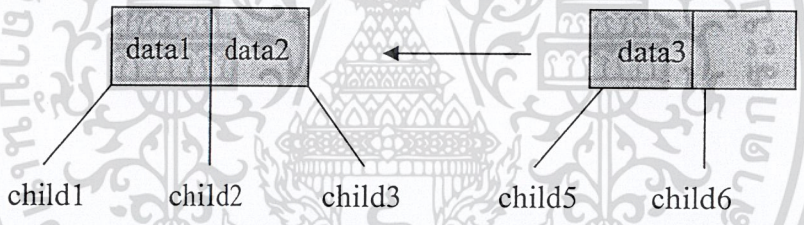


รูปที่ 2-5 แสดงการเก็บข้อมูลของ 2-3 tree (3)

หากการเพิ่มข้อมูลใหม่ทำให้มีการแตกตัวเกิดขึ้นที่ root จะทำให้มี root ใหม่ และ depth ของ tree จะเพิ่มขึ้น 1

2.2.3 กฎการแตกตัว

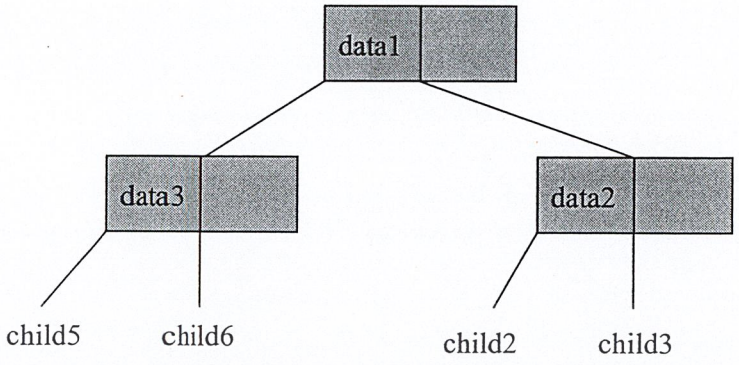
ในกรณีที่มีการแตกตัวของโหนดที่ไม่ใช่ leaf จำเป็นต้องคำนึงถึงโหนดลูกของโหนดที่มีการแตกตัวด้วย ซึ่งตำแหน่งของโหนดลูก หลังการแตกตัว จะเป็นไปตามกฎดังนี้



รูปที่ 2-6 แสดงกฎการแตกตัว (1)

สมมุติให้ [data1|data2] เป็นโหนดใน 2-3 tree และ [data3|] เป็นโหนดที่ต้องการเก็บใน [data1|data2]

Case 1 : child1 = data3



รูปที่ 2-7 แสดงกฎการแตกตัว (2)

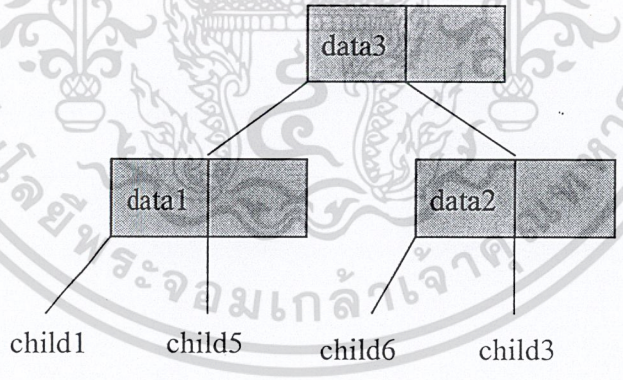
กรณีที่ 1 โหนด [data3|] ถูกเพิ่มระดับจาก child 1

data 1 จะเป็น key กลาง ได้รับการเพิ่มระดับ 1 ขั้น

data 3 จะเป็น node ลูกทางซ้ายของ data1 มีโหนดลูกเป็น child 5 และ child 6

data 2 จะเป็น node ลูกทางขวาของ data1 มีโหนดลูกเป็น child 2 และ child 3

Case 2 : child2 = data3



รูปที่ 2-8 แสดงกฎการแตกตัว (3)

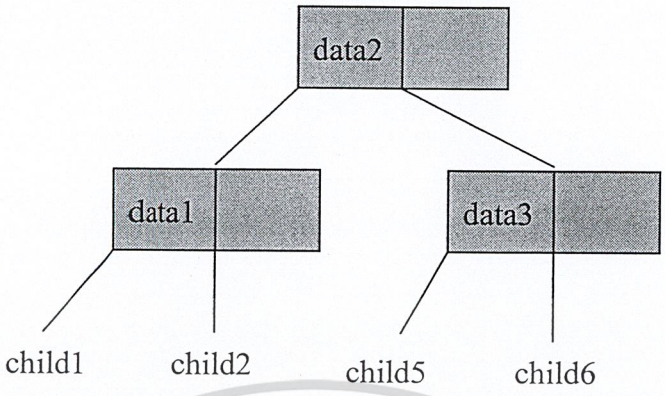
กรณีที่ 2 โหนด [data3|] ถูกเพิ่มระดับจาก child 2

data 3 จะเป็น key กลาง ได้รับการเพิ่มระดับ 1 ขั้น

data 1 จะเป็น node ลูกทางซ้ายของ data3 มีโหนดลูกเป็น child 1 และ child 5

data 2 จะเป็น node ลูกทางขวาของ data3 มีโหนดลูกเป็น child 6 และ child 3

Case 3 : child3 = data3



รูปที่ 2-9 แสดงกฎการแตกตัว (4)

กรณีที่ 3 โหนด [data3|] ถูกเพิ่มระดับจาก child 3

data 2 จะเป็น key กลาง ได้รับการเพิ่มระดับ 1 ขั้น

data 1 จะเป็น node ลูกทางซ้ายของ data2 มีโหนดลูกเป็น child 1 และ child 2

data 3 จะเป็น node ลูกทางขวาของ data2 มีโหนดลูกเป็น child 5 และ child 6

2.2.4 ความเร็ว , ข้อดี, ข้อเสีย

ความเร็วในการจัดเก็บข้อมูล 0.18 วินาที

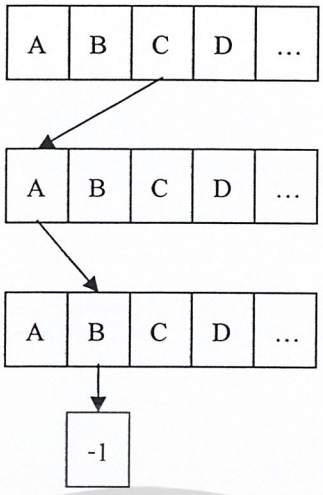
ความเร็วในการค้นหาคำ (แบบ Successful) 0.15 วินาที

ข้อดี ค้นหาข้อมูลได้รวดเร็ว เพราะ tree จะปรับตัวเองให้สมดุลตลอดเวลา, เหมาะกับข้อมูลที่ไม่มีการจัดเรียงและมีการจัดเรียงทุกรูปแบบ, ประหยัดเนื้อที่หน่วยความจำ

ข้อเสีย ในการกระทำใดๆกับข้อมูล 1 โหนด จะต้องกระทำกับข้อมูล 2 ชุด ทำให้ความเร็วในการทำงานลดลง, ไม่รองรับการค้นหาข้อมูลแบบ unsuccessful search, ในกรณีที่คำมีความยาวกว่าที่กำหนดไว้ จะไม่สามารถเก็บข้อมูลได้อย่างถูกต้อง

2.3 โครงสร้างข้อมูลแบบ access trie

โครงสร้างข้อมูลแบบ access trie เป็นโครงสร้างข้อมูลแบบ trie ที่ใช้อาร์เรย์ในการเก็บข้อมูลใน 1 โหนดอาร์เรย์ นี้จะมีจำนวน record เท่ากับหรือมากกว่าจำนวนสมาชิกของเซ็ตของตัวอักษร เช่น A-Z ก็จะมี 26 ตัวและใช้หมายเลขลำดับของ record ในการบอกตัวอักษร เช่น 1 = A , 2 = B ข้อมูลใน อาร์เรย์จะเก็บสถานะของคำว่าเป็นคำที่มีความหมายหรือไม่ ในที่นี้ผมใช้ -1 หมายถึงคำเป็นคำที่มีความหมายและ 0 เป็นคำที่ไม่มีความหมาย



รูปที่ 2-10 แสดงโครงสร้างของ access trie

2.3.1 การค้นหาข้อมูล

การค้นหาข้อมูลในโครงสร้างแบบ Access trie สามารถทำได้โดยเทียบคำที่ต้องการค้นหาทีละตัวอักษรเป็นตำแหน่งของข้อมูลเช่น คำว่า CAB จะมีการค้นหาดังนี้

ไปยัง record ที่ 3 และ move ไปยัง child

ไปยัง record ที่ 1 และ move ไปยัง child

ไปยัง record ที่ 2 จบสตริง อ่านค่า status ได้ -1 เป็นคำที่มีความหมาย

2.3.2 การเก็บข้อมูล

การเก็บข้อมูลใน Access trie สามารถทำได้โดยเทียบคำที่ต้องการค้นหาทีละตัวอักษรเป็นตำแหน่งของข้อมูลเช่น คำว่า CAB จะมีการเก็บข้อมูลดังนี้

ไปยัง record ที่ 3 ถ้ามี child ให้ move ไปยัง child ถ้าไม่มี child ให้สร้าง node ใหม่

ไปยัง record ที่ 1 ถ้ามี child ให้ move ไปยัง child ถ้าไม่มี child ให้สร้าง node ใหม่

ไปยัง record ที่ 2 จบสตริง บันทึกค่า status เป็น -1

2.3.3 ความเร็ว , ข้อดี, ข้อเสีย

ความเร็วในการจัดเก็บข้อมูล 0.751 วินาที

ความเร็วในการค้นหาคำ (แบบ Unsuccessful) 0.11 วินาที

ข้อดี มีความเร็วในการค้นหาข้อมูลสูง, รองรับการค้นหาคำแบบ Unsuccessful Search , ไม่จำกัดความยาวของคำ

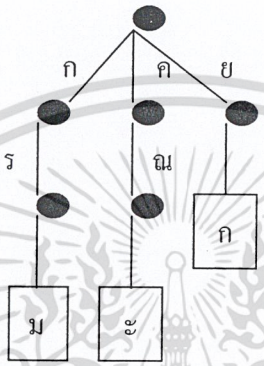
ข้อเสีย จำเป็นต้องใช้ใช้หน่วยความจำในปริมาณมหาศาล, มี record ที่ไม่ได้ใช้งานเป็นจำนวนมาก

Acess Trie เป็นโครงสร้างข้อมูลที่ทำงานได้รวดเร็ว มีขั้นตอนการทำงานที่เรียบง่ายและมีความสามารถตรงกับความต้องการในการนำมาใช้ในงานตัดคำแต่มีการใช้พื้นที่หน่วยความจำในปริมาณ

มหาศาล ดังนั้น จึงต้องหาวิธีพัฒนาโครงสร้างข้อมูล ให้มีความเร็วในการทำงานใกล้เคียงกับโครงสร้างข้อมูลแบบ Acess Trie และใช้เนื้อที่หน่วยความจำน้อยลง ดังจะได้เห็นในโครงสร้างข้อมูลแบบต่อไป

2.4 โครงสร้างข้อมูลแบบ trie

โครงสร้างข้อมูลแบบ trie ประกอบด้วยกลุ่มของ node ที่ต่อกันแบบ link list ในแต่ละ node จะมีข้อมูล และ ตัวชี้ 2 ตัว ตัวแรก สำหรับชี้ไปยังโหนดถัดไปที่อยู่ในระดับเดียวกัน และอีกตัวหนึ่งชี้ไปยังโหนด child ของมัน



รูปที่ 2-11 แสดงโครงสร้างข้อมูลแบบ trie

2.4.1 การค้นหาข้อมูล

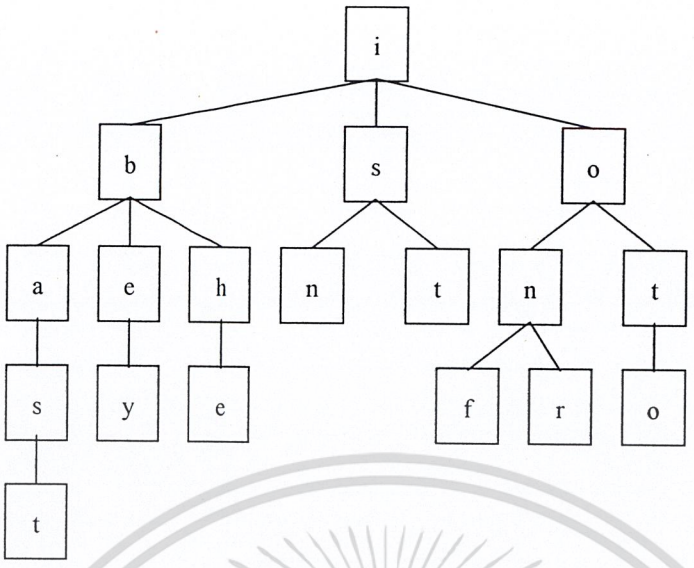
การค้นหาข้อมูลในโครงสร้างแบบ Trie จะเริ่มด้วยการเปรียบเทียบตัวอักษรตัวแรกของสตริงกับข้อมูลในโหนด root หากไม่ตรงก็จะไปค้นหาต่อในโหนดถัดไปที่อยู่ในระดับเดียวกันไปเรื่อยๆ เหมือนการค้นหาข้อมูลใน link list เมื่อพบโหนดที่มีข้อมูลตรงแล้วก็จะค้นหาตัวอักษรถัดไปในโหนด child เป็นเช่นนี้เรื่อยไปจนจบสตริง

2.4.2 การเก็บข้อมูล

ในการเก็บสตริง s ลงใน trie มีความเป็นไปได้ที่ส่วนต้นของสตริง s จะมีอยู่ใน trie อยู่แล้ว ดังนั้น การเก็บข้อมูลลงใน trie จะต้องค้นหาข้อมูลส่วนที่ซ้ำกับข้อมูลใน trie จนกระทั่งพบตำแหน่งที่ไม่มีอยู่ (ค่า null) แล้วจึงสร้างโหนดใหม่ที่เป็นข้อมูลส่วนปลายของสตริง s ต่อเข้าไปใน trie

2.5 โครงสร้างข้อมูลแบบ ternary tree

โครงสร้างข้อมูลแบบ ternary tree เป็นโครงสร้างข้อมูลที่ปรับปรุงขึ้นมาจาก trie โดยเปลี่ยนโครงสร้างของ node ที่อยู่ระดับเดียวกันจาก linear link list เป็น binary tree ทำให้ในแต่ละ node มีกิ่ง 3 กิ่ง คือ กิ่งซ้ายและขวาชี้ไปยังโหนดในระดับเดียวกัน โดยใช้โครงสร้างข้อมูลแบบ binary tree และ กิ่งกลาง ชี้ไปยังโหนด child ในลักษณะเดียวกับ trie ทำให้ ternary tree เป็นโครงสร้างข้อมูลที่รวมความสามารถในการอ่าน input ทีละตัวของโครงสร้างข้อมูลแบบ trie กับความเร็วในการค้นหาของ binary tree เข้าไว้ด้วยกัน



รูปที่ 2-12 แสดงโครงสร้างข้อมูลของ ternary tree

2.5.1 การค้นหาข้อมูล

การค้นหาข้อมูลในโครงสร้างแบบ Ternary Tree จะเริ่มด้วยการเปรียบเทียบตัวอักษรตัวแรกของสตริงกับข้อมูลในโหนด root หากไม่ตรงก็จะไปค้นหาต่อในโหนดถัดไปที่อยู่ในระดับเดียวกัน โดยจะค้นหาในกิ่งทางซ้ายถ้าข้อมูลมีค่าน้อยกว่า และค้นหาในกิ่งทางขวาถ้าข้อมูลมีค่ามากกว่าค่าในโหนด เมื่อพบโหนดที่มีข้อมูลตรงแล้วก็จะค้นหาตัวอักษรถัดไปในโหนด child เป็นเช่นนี้เรื่อยไปจนจบสตริง

2.5.2 การเก็บข้อมูล

ในการเก็บสตริง s ลงใน ternary tree มีความเป็นไปได้ที่ส่วนต้นของสตริง s จะมีอยู่ในโครงสร้างข้อมูลอยู่แล้ว ดังนั้น การเก็บข้อมูลลงใน ternary tree จะต้องค้นหาข้อมูลส่วนที่ซ้ำกับข้อมูลใน ternary tree จนกระทั่งพบตำแหน่งที่ไม่มีอยู่ (ค่า null) แล้วจึงสร้างโหนดใหม่ที่เป็นข้อมูลส่วนปลายของสตริง s ต่อเข้าไปในโครงสร้างข้อมูล

2.5.3 ความเร็ว , ข้อดี, ข้อเสีย

ข้อดี สามารถทำการค้นหาแบบตัวอักษรต่อตัวอักษรได้, รองรับการพัฒนาได้หลายรูปแบบ เช่น การค้นหาคำเป็นส่วนๆ หรือการค้นหาคำเหมือนคำใกล้เคียง, ง่ายต่อการประยุกต์ใช้งาน, เหมาะสำหรับงานที่มีอินพุตไม่แน่นอน

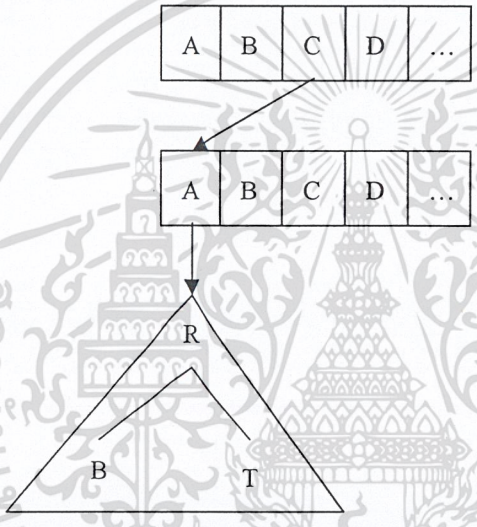
ข้อเสีย เมื่อมีอินพุตที่แน่นอนแล้ว ต้องมีการปรับเปลี่ยนการทำงานเพื่อเพิ่มความเร็วในการทำงาน, ไม่เหมาะสำหรับอินพุตที่เป็น order list

2.6 โครงสร้างข้อมูลแบบ burst trie

โครงสร้างข้อมูลแบบ burst trie เป็นโครงสร้างข้อมูลที่ประกอบด้วยโครงสร้างข้อมูล 2 ชนิด โดยมีแนวคิดว่า ในสตริงส่วนใหญ่ ตัวอักษรส่วนต้นของสตริงจะมีความหลากหลายสูงเช่น ตัวอักษรในตำแหน่งที่ 1 และ 2 จะสามารถเป็นไปได้ทุกตัวอักษร จึงน่าจะนำ Access trie มาใช้และในส่วนที่เหลือก็ใช้โครงสร้างข้อมูลแบบอื่นเพื่อให้ประหยัดพื้นที่หน่วยความจำ

โครงสร้างข้อมูลแบบ burst trie ประกอบด้วยส่วนประกอบ 3 ส่วนคือ

1. ส่วน access trie
2. ส่วน container
3. record



รูปที่ 2-13 แสดงโครงสร้างข้อมูลของ burst trie

ส่วน access trie จะเป็นส่วนที่มีความหลากหลายของตัวอักษร และเป็นทางผ่านในการค้นหาคำจำนวนมาก

ส่วน container จะเป็นส่วนที่เก็บ record เอาไว้ด้วยโครงสร้างข้อมูลอีกแบบหนึ่ง ซึ่งในที่นี้ใช้ ternary tree เพราะเป็นโครงสร้างข้อมูลที่มีประสิทธิภาพ และ container จะเก็บบันทึกจำนวนของ record ที่อยู่ภายในเอาไว้ด้วย หากจำนวนของ record เกินกว่าค่า Limit จะทำการเปลี่ยนสภาพ container ไปเป็น node ใหม่ ซึ่งเป็น Access trie โดยมี container เล็กๆ เชื่อมต่ออยู่กับ Access trie เรียกการเปลี่ยนสภาพนี้ว่าการ "burst"

2.6.1 การค้นหาข้อมูล

การค้นหาข้อมูลในโครงสร้างแบบ burst trie จะทำการค้นหาทีละตัวอักษร หากการค้นหาทำในส่วนของ Access trie ก็จะทำการเปรียบเทียบค่าเป็นหมายเลข record ใน อาร์เรย์ ตามแบบ Access trie หากการค้นหาทำในส่วนของ container ก็จะใช้วิธีการค้นหาตามแบบโครงสร้างข้อมูลที่ใช้ใน container

2.6.2 การเก็บข้อมูล

ในการเก็บสตริง s ลงใน burst trie มีความเป็นไปได้ที่ส่วนต้นของสตริง s จะมีอยู่ในโครงสร้างข้อมูลอยู่แล้ว ดังนั้น การเก็บข้อมูลลงใน burst trie จะต้องค้นหาข้อมูลส่วนที่ซ้ำกับข้อมูลใน burst trie จนกระทั่งพบตำแหน่งที่ไม่มีอยู่ (ค่า null) แล้วจึงสร้างโหนดใหม่ที่เป็นข้อมูลส่วนปลายของสตริง s ต่อเข้าไปในโครงสร้างข้อมูล หากข้อมูลถูกเก็บลงในส่วน container จะมีการเพิ่มค่าจำนวน record แล้วเปรียบเทียบกับค่า limit หากมีค่าเกินกว่าค่า limit จะต้องทำการ burst container ไปเป็น access trie

2.6.3 การ burst container ที่ใช้โครงสร้างข้อมูลแบบ Ternary tree

เริ่มต้น จะสร้าง node ใหม่เป็นชนิด acess trie แล้วเริ่มถ่ายโอนข้อมูลจาก root ของ container โดยจะนำข้อมูลใน node ไปเก็บใน Access trie แล้วนำ child ของ node ไปเป็น container หลังจากนั้นก็ถ่ายโอนข้อมูลในโหนดซ้ายและ โหนดขวาต่อไป

บทที่ 3
การประยุกต์ใช้โครงสร้างข้อมูลแบบต่างๆ ในการทำพจนานุกรมเพื่อการตัดคำ

3.1 Hash Data Structure

3.1.1 โครงสร้างข้อมูลในภาษาซี

```
char* data[memsize];
int max_rehash;
int numword=0;
```

ข้อมูลจะเก็บไว้ในอาเรย์ data[] โดยมีค่าคงที่ memsize เป็นตัวบอกจำนวนลำดับของอาเรย์ และมีตัวแปรเก็บสถานะของกลุ่มข้อมูลคือ max_rehash สำหรับเก็บจำนวนครั้งของการทำ rehash ที่มากที่สุดที่ปรากฏในการเก็บข้อมูล และตัวแปร numword สำหรับเก็บจำนวนคำที่มีอยู่ในกลุ่มข้อมูล

3.1.2 การค้นหาข้อมูลโครงสร้าง

เราจะนำคำที่ต้องการหามาประมวลผลใน hash function เพื่อหาตำแหน่งของข้อมูล หากข้อมูลที่อยู่ในตำแหน่งนั้นๆ ไม่ใช่ข้อมูลที่ต้องการหา ก็จะเพิ่มค่าในตัวแปร rh แล้วทำการ hash ใหม่ ซึ่งจะทำเช่นนี้ไปเรื่อยๆ จนกว่าจะพบข้อมูล สำหรับในกรณีที่ ไม่มีข้อมูลอยู่ในกลุ่มข้อมูลนั้น ค่า output ที่ได้จาก hash function จะชี้ไปยังช่องเก็บข้อมูลที่ว่างเปล่า หรือ ค่า rh จะถูกเพิ่มไปเรื่อยๆจนกระทั่งมีค่าเกินกว่าตัวแปร max_rehash

Code ข้างล่างเป็นฟังก์ชั่นค้นหาข้อมูลในโปรแกรม ในกรณีที่พบข้อมูลจะ return ลำดับของข้อมูลและจะ return memsize+1 หากไม่พบข้อมูล

```
int find(char* word)
{
        int rh=0;
        // first step, hash word with seed=0 , x is array number
        int x=hash(word,rh);
        // while not found and can be rehash
        while (data[x]!=NULL&&strcmp(word,data[x])!=0&&rh<max_rehash)
        {
                rh++;                   // increse seed value
                x=hash(word,rh);        // rehash
```

```
        }
        // finished hashing now, if found return x, else return memsize+1
        if (data[x]!=NULL&&strcmp(word,data[x])==0)                    {
           return x;
        } else
        {
          return memsize+1;
        }
}
```

3.1.3 การ insert ข้อมูล

ในตอนเริ่มต้น จะหาค่า hash โดยกำหนด seed เป็น 0 หากตำแหน่งที่ได้ไม่มีข้อมูลเก็บอยู่ ก็จะใส่ข้อมูลลงไปในตำแหน่งนั้น มิเช่นนั้นก็จะเพิ่มค่า seed แล้วทำการ rehash จนกว่าจะพบตำแหน่งที่ว่าง หากมีการ rehash เกินกว่าครั้งที่มากที่สุดที่เคยทำ ก็จะบันทึกจำนวนครั้งลงในตัวแปร max_rehash

```
void insert(char* word)
{
if (*word!='\0')
{
        int hashres=hash(word,0);
        int done=0;
        int rh=0;
        while (done==0)
        {
                if (data[hashres]==NULL)
                {
                        data[hashres]=word;
                        done=1;
                        numword++;
                } else
                {
                    rh++;
                    rehash_count++;
                    if (rh>max_rehash) max_rehash=rh;
```

```
                    hashres=hash(word,rh);
                }
        }
}
}
```

3.1.4 ประสิทธิภาพ (ความเร็วและเนื้อที่ที่ต้องใช้ในการทดสอบ)

ในการทดสอบประสิทธิภาพของโครงสร้างข้อมูลแบบ hash จะเป็นการ insert คำภาษาไทย จำนวน 34022 คำ ซึ่งมีความยาวในแต่ละคำไม่เกิน 40 ตัวอักษร แล้วทำการวัดความเร็วและเนื้อที่ที่ใช้ไป และทำการค้นหาคำ 1 รอบเพื่อทำการวัดความเร็วในการค้นหา และบันทึกไว้เพื่อนำไปเปรียบเทียบ กับโครงสร้างข้อมูลแบบอื่นๆ

เนื้อที่ที่ใช้ในการเก็บข้อมูล 32,000,000 ไบต์ (hash อาเรย์ขนาด 200000 record)
ความเร็วในการจัดเก็บข้อมูล 0.07 วินาที
ความเร็วในการค้นหาคำ (แบบ Successful) 0.21 วินาที
มีการ rehash ในการจัดเก็บข้อมูลทั้งหมด 3502 ครั้ง
คำที่มีการ rehash มากที่สุด 4 ครั้ง

3.1.5 การทำงานของแต่ละฟังก์ชั่น

3.1.5.1 ฟังก์ชั่น -> int strcmp(char *s1, char *s2)
พารามิเตอร์ -> s1,s2 เป็น String ที่มีการประกาศตัวแปรแบบ pointer
ส่งค่ากลับ -> เป็น integer หาก s1 < s2 จะมีค่าน้อยกว่า 0 หาก s1=s2 จะมีค่าเป็น 0 หาก s1>s2 จะมี่มากกว่า 0
จุดประสงค์ -> เพื่อเปรียบเทียบลำดับก่อนหลังของคำ 2 คำ
เรียกใช้งาน ->1. เรียกจากฟังก์ชั่น find()

3.1.5.2 ฟังก์ชั่น -> unsigned int hash(char *word,unsigned int seed)
พารามิเตอร์ -> word เป็น String ที่มีการประกาศตัวแปรแบบ pointer
seed เป็นจำนวนครั้งของการ rehash
ส่งค่ากลับ -> เป็น integer แบบไม่คิดเครื่องหมาย มีค่าเป็น index ของ record ใน array
จุดประสงค์ -> เพื่อหาค่าที่ได้จากกระบวนการ hash
เรียกใช้งาน -> 1. จากฟังก์ชั่น find()
2. จากฟังก์ชั่น insert()

3.1.5.3 ฟังก์ชั่น -> void insert(char* word)

พารามิเตอร์ -> word เป็น String ที่มีการประกาศตัวแปรแบบ pointer

ส่งค่ากลับ -> ไม่มี

จุดประสงค์ -> เพื่อเพิ่มคำลงในโครงสร้างข้อมูล

เรียกใช้งาน -> 1. จากฟังก์ชั่น readinput()

3.1.5.4 ฟังก์ชั่น -> void readinput()

พารามิเตอร์ -> ไม่มี

ส่งค่ากลับ -> ไม่มี

จุดประสงค์ -> เพื่อรับ input จากไฟล์ แล้วนำไปใส่ในโครงสร้างข้อมูล

เรียกใช้งาน -> 1. จากฟังก์ชั่น main()

3.1.5.5 ฟังก์ชั่น -> int find(char* word)

พารามิเตอร์ -> word เป็น String ที่มีการประกาศตัวแปรแบบ pointer

ส่งค่ากลับ -> เป็น integer แบบไม่คิดเครื่องหมาย มีค่าเป็น index ของ record ใน array

จุดประสงค์ -> เพื่อหาตำแหน่งของคำในในโครงสร้างข้อมูล

เรียกใช้งาน -> 1.จากฟังก์ชั่น main()

3.1.5.6 ฟังก์ชั่น -> void main()

พารามิเตอร์ -> ไม่มี

ส่งค่ากลับ -> ไม่มี

จุดประสงค์ -> 1. สร้างโครงสร้างข้อมูล

2. อ่าน input จากไฟล์และค้นหาข้อมูล

3. จับเวลาการทำงานและรายงานผล

3.1.6 สรุปข้อดีข้อเสีย

ข้อดี

1. มีความเร็วในการทำงานสูง
2. ง่ายในการประยุกต์ใช้งานหรือแก้ไขเพิ่มเติม
3. ใช้เนื้อที่หน่วยความจำในปริมาณคงที่ และสามารถจำกัดขอบเขตการใช้หน่วยความจำได้
4. เหมาะกับ input ทั้งแบบ ordered list , unordered list และ reverse ordered list

ข้อเสีย

1. ไม่เหมาะสำหรับการค้นหาแบบรับ input ทีละตัวอักษร

2. ใช้หน่วยความจำค่อนข้างมากและสิ้นเปลืองไปกับ record ที่ไม่มีข้อมูลมาก
3. ไม่รองรับการค้นหาคำเหมือนคำใกล้เคียง
4. ในกรณีที่คำมีความยาวกว่าที่กำหนดไว้ จะไม่สามารถเก็บข้อมูลได้อย่างถูกต้อง

3.2 2-3 Tree Data Structure

3.2.1 โครงสร้างข้อมูลในภาษาซี

```
struct t23
{
 char data[2][maxchar];
 t23_ptr child[3];
};
```

ข้อมูลจะถูกเก็บไว้ในตัวแปร data[] ซึ่งจะมีข้อมูลได้ 2 ชุดใน 1 โหนด และมีตัวแปร child[] สำหรับชี้ไปยังโหนดถัดไป

3.2.2 การค้นหาข้อมูลโครงสร้าง

เราจะใช้ recursive ในการค้นหา จะทำการ return พอยเตอร์ของ node ที่เก็บข้อมูลหากไม่พบก็จะ return null

ในตอนเริ่มต้นจะทำการเปรียบเทียบ input กับข้อมูลทั้งสอง หากพบก็จะ return พอยเตอร์ที่ชี้ตัวเอง หากไม่พบก็จะหาต่อใน node ถัดไปที่เหมาะสมกับผลการเปรียบเทียบ

```
t23_ptr find(t23_ptr a,char* w)
{
  if (a!=NULL)
 {
            int res1=strcmp(w,a->data[0]);
            int res2=strcmp(w,a->data[1]);
            int av=strcmp(a->data[1],"")==0;
            if (res1<0) {return find(a->child[0],w);}
            if (res1==0) {return a;}
            if (res1>0&&(res2<0||av)) {return find(a->child[1],w);}
            if (res2==0) {return a;}
```

```
            if (res2>0&&!av) {return find(a->child[2],w);}
    } else {
     return NULL;
    }
}
```

3.2.3 การ insert ข้อมูล

ฟังก์ชั่น insert จะทำการ insert ข้อมูลลงไปใน tree และจะไม่ทำอะไรถ้ามีข้อมูลนั้นอยู่ใน tree อยู่แล้ว และหากข้อมูลที่ insert ลงไปทำให้เกิด การ split ก็จะ return root อันใหม่ออกมา หากการ insert ไม่ทำให้เกิดการ split ก็จะ return null

การเรียกใช้ฟังก์ชั่นทำได้โดยคำสั่ง

```
tmp=insert(root,newdata);
if (tmp!=NULL) {root=tmp;}
```

การนำมาประยุกต์ใช้ทำในลักษณะ recursive โดย root ของ tree ที่ต้องการ insert จะอยู่ในตัวแปร a และข้อมูลใหม่จะอยู่ในตัวแปร c หาก node a ไม่ใช่ leaf ก็จะ insert ไปยัง child ที่เหมาะสม เมื่อ a เป็น leaf จึงทำการเพิ่มข้อมูลลงใน tree หาก node a มีข้อมูลอยู่เต็มอยู่แล้ว ก็จะทำการ split node a ออกเป็น 3 node แล้วนำ key กลางมาเป็น root ใหม่ และ return ออกไป

ในกรณีที่ node a ไม่ใช่ root แต่การ insert ข้อมูลลงใน child ได้รับค่าที่ไม่เป็น null กลับมา node a จะต้องนำโหนดที่ถูก return กลับมาใส่ใน data หากมีข้อมูลเต็มอยู่แล้วก็จะเอา node a ออกเป็น 3 node แล้วนำ key กลางมาเป็น root ใหม่ และ return ออกไปอีกทอดหนึ่ง

```
t23_ptr insert(t23_ptr a,t23_ptr c)
{
   t23_ptr re=NULL; // re is return value

   if (a!=NULL) { // if a is not null
            t23_ptr b=NULL; // b is result from inserting data to child
            /*insert to true child */
            int res1=strcmp(c->data[0],a->data[0]);
            int res2=strcmp(c->data[0],a->data[1]);
```

```
int dir=0;
int av=strcmp(a->data[1],"")==0;
int blank=strcmp(a->data[0],"")==0;

if (res1<0||blank) {dir=0;b=insert(a->child[0],c);}
if (res1==0) {/* already have this data, do nothing*/}
if (res1>0&&!blank&&(res2<0||av)) {dir=1;b=insert(a->child[1],c);}
if (res2==0&&!av) {/* already have this data, do nothing*/}
if (res2>0&&!av) {dir=2;b=insert(a->child[2],c);}

/* if it's return unnull , insert here */
if (b!=NULL) {
 if (dir==0)
 {
        if (av) // have space , insert here
        {
          //move old data(data[0]) to new place(data [1])
          strcpy(a->data[1],a->data[0]);
          a->child[2]=a->child[1];
          //insert new data
          strcpy(a->data[0],b->data[0]);
          a->child[0]=b->child[0];
          a->child[1]=b->child[1];
          //can insert , return null
          re=NULL;
          delete b;
        } else{ // No space, must split a
          // splite a to a1,a2
          t23_ptr a1=newnode("");
          t23_ptr a2=newnode("");
          strcpy(a1->data[0],a->data[0]);
          strcpy(a2->data[0],a->data[1]);
          a1->child[0]=b;
```

```
            a1->child[1]=a2;
            a2->child[0]=a->child[1];
            a2->child[1]=a->child[2];
            re=a1;
            delete a;
        }
    }
    if (dir==1)
    {
        if (av) // have space , insert here
        {
            strcpy(a->data[1],b->data[0]);
            a->child[1]=b->child[0];
            a->child[2]=b->child[1];
            // can insert , return null
            re=NULL;
            delete b;
        } else{ // No space, must split a
            // splite a to a1,a2
            t23_ptr a1=newnode("");
            t23_ptr a2=newnode("");
            strcpy(a1->data[0],a->data[0]);
            strcpy(a2->data[0],a->data[1]);
            a1->child[0]=a->child[0];
            a1->child[1]=b->child[0];
            a2->child[0]=b->child[1];
            a2->child[1]=a->child[2];
            b->child[0]=a1;
            b->child[1]=a2;
            re=b;
            delete a;
        }
    }
    if (dir==2)
```

```
            {
                // splite a to a1,a2
                t23_ptr a1=newnode("");
                t23_ptr a2=newnode("");
                strcpy(a1->data[0],a->data[0]);
                strcpy(a2->data[0],a->data[1]);
                a1->child[0]=a->child[0];
                a1->child[1]=a->child[1];
                a2->child[0]=a1;
                a2->child[1]=b;
                re=a2;
                delete a;
            }
          }
        } else { // if a is NULL , Return c
          re=c;
        }
        return re;
}
```

3.2.4 ประสิทธิภาพ (ความเร็วและเนื้อที่ที่ต้องใช้ในการทดสอบ)

เนื้อที่ที่ต้องใช้ในการเก็บข้อมูล 3,129,380 byte

ความเร็วในการจัดเก็บข้อมูล 0.18 วินาที

ความเร็วในการค้นหาคำ (แบบ Successful) 0.15 วินาที

3.2.5 การทำงานของแต่ละฟังก์ชั่น

3.2.5.1 ฟังก์ชั่น -> t23_ptr newnode(char* a)

พารามิเตอร์ -> a เป็น String ที่มีการประกาศตัวแปรแบบ pointer

ส่งค่ากลับ -> เป็น pointer ของ node ใหม่

จุดประสงค์ ->เพื่อสร้างโหนดใหม่ จับจองเนื้อที่ในเมมโมรี่ และ กำหนดค่าเริ่มต้น

เรียกใช้งาน -> 1. จากฟังก์ชั่น insert()

2. จากฟังก์ชั่น main()

3.2.5.2 ฟังก์ชั่น -> t23_ptr insert(t23_ptr a,t23_ptr c)

พารามิเตอร์ -> 1. a เป็น root ของ tree ที่ต้องการจะเพิ่มคำ

2. c เป็น pointer ของ ข้อมูลใหม่

ส่งค่ากลับ -> หากมีการ splite จะเป็น เป็น pointer ของ node ใหม่ หากไม่มีการ splite จะเป็น null

จุดประสงค์ -> เพื่อเพิ่มข้อมูลลงใน tree

เรียกใช้งาน -> 1. จากตัวเอง

2. จากฟังก์ชั่น main()

3.2.5.3 ฟังก์ชั่น -> t23_ptr find(t23_ptr a,char* w)

พารามิเตอร์ -> 1. a เป็น root ของ tree ที่ต้องการจะค้นหาคำ

2. w เป็นคำที่ต้องการจะค้นหา

ส่งค่ากลับ -> เป็น pointer ของโหนดที่เก็บคำ input อยู่

จุดประสงค์ -> เพื่อค้นหาคำใน tree, subtree

เรียกใช้งาน -> 1. จากตัวมันเอง

2. จากฟังก์ชั่น main()

3.2.5.4 ฟังก์ชั่น ->traverse(t23_ptr a)

พารามิเตอร์ ->a เป็น root ของ tree ที่ต้องการจะ traverse

ส่งค่ากลับ -> ไม่มี

จุดประสงค์ -> เพื่อแสดงคำทั้งหมดใน tree โดย traverse แบบ inorder list

เรียกใช้งาน -> 1. จากตัวมันเอง

2. จากฟังก์ชั่น main()

3.2.5.5 ฟังก์ชั่น -> long size(t23_ptr a)

พารามิเตอร์ ->a เป็น root ของ tree ที่ต้องการจะวัดขนาด

ส่งค่ากลับ -> เป็น longint มีค่าเป็นขนาดของ หน่วยความจำที่ใช้ในการสร้าง tree

จุดประสงค์ -> เพื่อหาขนาดของหน่วยความจำที่ใช้ในการสร้าง tree

เรียกใช้งาน -> 1. จากตัวมันเอง

2. จากฟังก์ชั่น main()

3.2.6 สรุปข้อดีข้อเสีย

ข้อดี

1. ค้นหาข้อมูลได้รวดเร็ว เพราะ tree จะปรับตัวเองให้สมดุลตลอดเวลา

2. เหมาะกับข้อมูลที่ไม่มีการจัดเรียงและมีการจัดเรียงทุกรูปแบบ
3. ประหยัดเนื้อที่หน่วยความจำ

ข้อเสีย

1. ในการกระทำใดๆกับข้อมูล 1 โหนด จะต้องกระทำกับข้อมูล 2 ชุด ทำให้ความเร็วในการทำงานลดลง
2. ไม่รองรับการค้นหาข้อมูลแบบ unsuccessful search
3. ในกรณีที่คำมีความยาวกว่าที่กำหนดไว้ จะไม่สามารถเก็บข้อมูลได้อย่างถูกต้อง

3.3 Access Trie Data Structure

3.3.1 โครงสร้างข้อมูลในภาษาซี

```
struct Anode
{
  char st[256];
  Aptr ptr[256];
}
```

ในแต่ละโหนดจะเป็น array มีจำนวน record เท่ากับจำนวนตัวอักษร คือ 256 record โดยมี st เก็บสถานะของคำ และ ptr สำหรับชี้โหนดถัดไป

3.3.2 การค้นหาข้อมูลโครงสร้าง

เราจะใช้ recursive ในการค้นหา จะทำการ return ค่า -1 หากพบข้อมูล และ return 0 หากไม่พบข้อมูล

```
int find(Aptr p,unsigned char *s)
{
    if (!p) return 0;
            unsigned char* q=s;
            ++q;
    if (*q==0x00)
    {
            return p->st[*s];
    } else {
```

```
    return find(p->ptr[*s],q);
   }
}
```

3.3.3 การ insert ข้อมูล

ฟังก์ชั่น insert จะทำการ insert ข้อมูลลงไปใน Access trie และจะไม่ทำอะไรถ้ามีข้อมูลนั้นอยู่แล้ว การเรียกใช้ฟังก์ชั่นทำได้โดยคำสั่ง

```
root=insert(root,newdata);
```

ในการนำมาประยุกต์ใช้หาก a เป็น null ก็จะสร้างโหนดใหม่ขึ้นมาแล้วใส่สถานะลงไป หากเป็นคำที่มีความหมาย สถานะจะมีค่าเป็น -1 หากไม่ใช่ หรือยังไม่จบคำ ก็จะเรียกตัวเองขึ้นมาทำงานเพื่อ insert ตัวอักษรถัดไปลงใน node ถัดไป

```
Aptr insert(Aptr a,unsigned char* w)
{
  unsigned char* t=w;
  if (!a)
  {
     a=init();
  }
  ++t;
  if (*t!=0x00)
  {
   a->ptr[*w]=insert(a->ptr[*w],t);
  } else {
   a->st[*w]=-1;
  }
  return a;
}
```

3.3.4 ประสิทธิภาพ (ความเร็วและเนื้อที่ที่ต้องใช้ในการทดสอบ)

เนื้อที่ที่ต้องใช้ในการเก็บข้อมูล 105,930,240 byte

ความเร็วในการจัดเก็บข้อมูล 0.751 วินาที

ความเร็วในการค้นหาคำ (แบบ Unsuccessful) 0.11 วินาที

3.3.5 หน้าที่ของแต่ละฟังก์ชั่น

3.3.5.1 ฟังก์ชั่น -> Aptr init()

พารามิเตอร์ -> ไม่มี

ส่งค่ากลับ -> เป็น pointer ของโหนดใหม่

จุดประสงค์ -> เพื่อสร้างโหนดใหม่และกำหนดค่าเริ่มต้น

เรียกใช้งาน -> 1. เรียกจากฟังก์ชั่น insert()

3.3.5.2 ฟังก์ชั่น -> Aptr insert(Aptr a,unsigned char* w)

พารามิเตอร์ -> 1. a เป็น pointer ของ trie ที่ต้องการเพิ่มข้อมูล

2. w เป็น String ที่จะเพิ่มลงไปในโครงสร้างข้อมูล

ส่งค่ากลับ -> เป็น pointer ของ root ของโครงสร้างข้อมูลที่เพิ่มข้อมูลแล้ว

จุดประสงค์ -> เพื่อเพิ่มข้อมูลลงในโครงสร้างข้อมูล

เรียกใช้งาน -> 1. จาก main() 2. จากตัวมันเอง

3.3.5.3 ฟังก์ชั่น -> int find(Aptr p,unsigned char *s)

พารามิเตอร์ -> 1. p เป็น pointer ของ trie ที่ต้องการค้นหาข้อมูล

2. s เป็น String ที่ต้องการค้นหา

ส่งค่ากลับ -> เป็นค่าสถานะของคำที่ต้องการค้นหา หากเป็น -1 แสดงว่ามีคำๆนั้นอยู่

จุดประสงค์ -> เพื่อตรวจสอบว่ามีคำอยู่ในโครงสร้างข้อมูลหรือไม่

เรียกใช้งาน -> 1. จาก main() 2. จากตัวมันเอง

3.3.5.4 ฟังก์ชั่น -> void subfix(Aptr p)

พารามิเตอร์ -> p เป็น pointer ของ subtrie ที่ต้องการแสดงข้อมูล

ส่งค่ากลับ -> ไม่มี

จุดประสงค์ -> เพื่อแสดงคำทั้งหมดที่เก็บอยู่ใน subtrie

เรียกใช้งาน -> 1. จาก mian() 2. จากตัวมันเอง

3.3.5.5 ฟังก์ชั่น -> long size(Aptr p)

พารามิเตอร์ -> p เป็น pointer ของ trie ที่ต้องการแสดงข้อมูล

ส่งค่ากลับ -> เป็นขนาดของหน่วยความจำที่ใช้ในการสร้าง trie

จุดประสงค์ -> เพื่อวัดขนาดของหน่วยความจำที่ใช้ในการสร้าง trie

เรียกใช้งาน -> 1. จาก mian() 2. จากตัวมันเอง

3.3.5.6 ฟังก์ชั่น -> main()

พารามิเตอร์ -> ไม่มี

ส่งค่ากลับ -> ไม่มี

จุดประสงค์ -> 1. เพื่ออ่านไฟล์ dictionary มาใส่ในโครงสร้างข้อมูล

2. เพื่ออ่านไฟล์ input แล้วทำการค้นหาข้อมูล

3. เพื่อวัดประสิทธิภาพของโครงสร้างข้อมูล โดยวัดขนาด,ความเร็วในการสร้างและค้นหาข้อมูล

3.3.6 สรุปข้อดีข้อเสีย

ข้อดี

1. มีความเร็วในการค้นหาข้อมูลสูง
2. รองรับการค้นหาคำแบบ Unsuccessful Search
3. ไม่จำกัดความยาวของคำ

ข้อเสีย

1. จำเป็นต้องใช้ใช้หน่วยความจำในปริมาณมหาศาล
2. มี record ที่ไม่ได้ใช้งานเป็นจำนวนมาก

3.4 Trie Data Structure

3.4.1 โครงสร้างข้อมูลในภาษาซี

```
typedef struct Tnode* Tptr;
struct Tnode
{
 char data;
 int st;
 Tptr child,next;
};
```

ข้อมูลจำนวน 1 ตัวอักษรจะเก็บไว้ในฟิลด์ data โดยมีฟิลด์ child สำหรับชี้โหนดของตัวอักษรต่อจาก data และ next สำหรับชี้โหนดถัดไปในลักษณะ link list ฟิลด์ st ทำหน้าที่บอกสถานะของโหนดว่า

เป็นโหนดที่มีความหมายหรือไม่ หากเป็น 0 แสดงว่าตำแหน่งของตัวอักษรในโหนดไม่ใช่ตำแหน่งจบคำ และหากเป็น -1 แสดงว่าเป็นตำแหน่งที่จบคำ เป็นคำที่มีความหมาย

3.4.2 การค้นหาข้อมูลโครงสร้าง

โปรแกรมจะเริ่มค้นหาที่โหนด root เป็นอันดับแรก หากเป็น Anode ก็จะทราบตำแหน่งของข้อมูลได้ทันทีจากลำดับของตัวอักษรและสามารถค้นหาข้อมูลส่วนที่เหลือใน subtrie ได้เลย แต่ในกรณีที่เป็น Ternary tree ก็จะค้นหาโดยใช้ฟังก์ชั่นของ Ternary tree

ในการเขียนโปรแกรมได้แบ่งการค้นหาเป็น walk() และ find() โดยฟังก์ชั่น walk() ทำหน้าที่ค้นหาโหนดที่มีข้อมูลตรงกับตัวอักษร ซึ่งจะถูกเรียกใช้จากฟังก์ชั่น find() ในการค้นหาตัวอักษรตามความยาวของ String

```
Tptr walk(Tptr p,char s)
{
    if (!p) return NULL;
    if (s == p->data)
                {return p;}
          else
          {return walk(p->next,s);}
}
Tptr find(Tptr p,char *s)
{
 Tptr o=p;
 while (*s&&o)
 {
   o=walk(o,*s);
   if (strlen(s)>1&&o) o=o->child;
   s++;
 }
 return o;
}
```


3.4.3 การ insert ข้อมูล

ฟังก์ชั่น insert ทำการ insert สตริงเข้าไปใน trie และจะไม่มีการ insert ถ้ามีข้อมูลอยู่แล้ว จะทำการ insert สตริง s ด้วย code root=insert(root,s)

```
Tptr insert(Tptr p, char *s)
{ if (p == NULL) {
        p = new Tnode;
        p->data = *s;
        p->st=0;
        p->next = p->child = NULL;
   }
   if (*s != p->data)
        p->next = insert(p->next, s);
   else if (*s == p->data) {
        if (strlen(s)>1)
            {p->child = insert(p->child, ++s);}
        else {p->st=-1;}
   }
   return p;
}
```

3.4.4 ประสิทธิภาพ (ความเร็วและเนื้อที่ที่ต้องใช้ในการทดสอบ)

ในการทดสอบประสิทธิภาพของโครงสร้างข้อมูลแบบ trie จะเป็นการ insert คำภาษาไทย จำนวน 34022 คำ ซึ่งมีความยาวในแต่ละคำไม่เกิน 40 ตัวอักษร แล้วทำการวัดความเร็วและเนื้อที่ที่ใช้ไป และทำการค้นหาคำ 1 รอบ เพื่อทำการวัดความเร็วในการค้นหา และบันทึกไว้เพื่อนำไปเปรียบเทียบ กับโครงสร้างข้อมูลแบบอื่นๆ

เนื้อที่ที่ใช้ในการเก็บข้อมูล 1,770,416 ไบต์
ความเร็วในการจัดเก็บข้อมูล 0.21 วินาที
ความเร็วในการค้นหาคำ (แบบ Successful) 0.17 วินาที
มีการประกาศใช้โหนดในการเก็บข้อมูลจำนวน 110651 โหนด

3.4.5 การทำงานของแต่ละฟังก์ชั่น

3.4.5.1 ฟังก์ชั่น -> Tptr insert(Tptr p, char *s)

พารามิเตอร์ -> 1. p เป็น pointer ของ Trie ที่จะเก็บข้อมูล

2. s เป็นสตริงที่จะเก็บใน p

จุดประสงค์ -> เพื่อเก็บสตริงใหม่ลงใน trie

เรียกใช้งาน ->1. เรียกจากฟังก์ชั่น main()

2. เรียกจากตัวเอง

3.4.5.2 ฟังก์ชั่น -> void insert(char* word)

พารามิเตอร์ -> word เป็น String ที่มีการประกาศตัวแปรแบบ pointer

ส่งค่ากลับ -> ไม่มี

จุดประสงค์ -> เพื่อเพิ่มคำลงในโครงสร้างข้อมูล

เรียกใช้งาน -> 1. จากฟังก์ชั่น readinput()

3.4.5.3 ฟังก์ชั่น -> void subfix(Tptr p)

พารามิเตอร์ -> p เป็น pointer ของ trie

ส่งค่ากลับ -> ไม่มี

จุดประสงค์ -> เพื่อแสดงสตริงที่บรรจุอยู่ใน trie

เรียกใช้งาน -> 1. จากฟังก์ชั่น main()

3.4.5.4 ฟังก์ชั่น -> Tptr walk(Tptr p,char s)

พารามิเตอร์ -> 1. p เป็น pointer ของ Trie

2. s เป็นตัวอักษรที่จะค้นหาใน p

ส่งค่ากลับ -> เป็น pointer ของ trie ที่มีข้อมูลตรงกับ s

จุดประสงค์ -> เพื่อหาโหนดที่เก็บตัวอักษร s ที่อยู่ในระดับความลึกเท่ากับ p

เรียกใช้งาน -> 1.จากฟังก์ชั่น find()

3.4.5.5 ฟังก์ชั่น -> Tptr find(Tptr p,char *s)

พารามิเตอร์ -> 1. p เป็น pointer ของ Trie

2. s เป็นสตริงที่จะค้นหาใน p

ส่งค่ากลับ -> เป็น pointer ของ trie ที่เก็บตัวอักษรสุดท้ายใน s

จุดประสงค์ -> เพื่อหาตำแหน่งของคำในในโครงสร้างข้อมูล

เรียกใช้งาน -> 1.จากฟังก์ชั่น main()

3.4.5.6 ฟังก์ชั่น -> void main()

พารามิเตอร์ -> ไม่มี

ส่งค่ากลับ -> ไม่มี

จุดประสงค์ -> 1. สร้างโครงสร้างข้อมูล

2. อ่าน input จากไฟล์และค้นหาข้อมูล

3. จับเวลาการทำงานและรายงานผล

3.4.6 สรุปข้อดีข้อเสีย

ข้อดี

1. มีความเร็วในการค้นหาข้อมูลสูง
2. ง่ายในการประยุกต์ใช้งานหรือแก้ไขเพิ่มเติม
3. ประหยัดหน่วยความจำ
4. เหมาะกับงานที่มี input ไม่แน่นอน
5. สามารถค้นหาคำทีละตัวอักษรได้

ข้อเสีย

1. มีโครงสร้างข้อมูลแบบ link list ทำให้ความเร็วในการค้นหาข้อมูลลดลง

3.5 Ternary Tree Data Structure

3.5.1 โครงสร้างข้อมูลในภาษาซี

```
typedef struct Tnode* Tptr;
struct Tnode
{
  char data;
  int st;
  Tptr hi,lo,eq;
};
```

ข้อมูลจำนวน 1 ตัวอักษรจะเก็บไว้ในฟิลด์ data โดยพ้อยเตอร์ 3 อันจะแสดงลูกได้ 3 ตัว ฟิลด์ st ทำหน้าที่บอกสถานะของโหนดว่าเป็นโหนดที่มีความหมายหรือไม่ หากเป็น 0 แสดงว่าตำแหน่งของตัวอักษรในโหนดไม่ใช่ตำแหน่งจบคำ และหากเป็น -1 แสดงว่าเป็นตำแหน่งที่จบคำ เป็นคำที่มีความหมาย

3.5.2 การค้นหาข้อมูลโครงสร้าง

การค้นหาจะกระทำทีละตัวอักษร โปรแกรมจะเริ่มค้นหาที่โหนด root เป็นอันดับแรก ด้วยวิธีการค้นหาแบบเดียวกับ binary search tree แสดงการทำงานโดยฟังก์ชั่น walk() เมื่อพบตัวอักษรนั้นแล้วจึงทำการค้นหาตัวอักษรตัวต่อไปในโหนด eq โดยมีการทำงานในลักษณะลูป ซึ่งแสดงการทำงานโดยฟังก์ชั่น find()

```
Tptr walk(Tptr p,char s)
{
    if (!p) return NULL;
    if (s == p->data)
            return p;
    if (s > p->data)
            return walk(p->hi,s);
    if (s < p->data)
            return walk(p->lo,s);
}

// find word function
Tptr find(Tptr p,char *s)
{
 Tptr o=p;
 while (*s&&o)
 {
   o=walk(o,*s);
   if (strlen(s)>1&&o) o=o->eq;
   s++;
 }
 return o;
}
```

3.5.3 การ insert ข้อมูล

ฟังก์ชั่น insert ทำการ insert สตริงเข้าไปใน trie และจะไม่มีการ insert ถ้ามีข้อมูลอยู่แล้วจะทำการ insert สตริง s ด้วย code root=insert(root,s)


```
Tptr insert(Tptr p, char *s)
```

```
{ if (p == NULL) {
        p = new Tnode;
        p->data = *s;
        p->st=0;
        p->lo = p->eq = p->hi = NULL;
    }
    if (*s < p->data)
        p->lo = insert(p->lo, s);
    else if (*s == p->data) {
        if (strlen(s)>1)
            {p->eq = insert(p->eq, ++s);}
        else {p->st=-1;}
    } else
        p->hi = insert(p->hi, s);
    return p;
}
```

3.5.4 ประสิทธิภาพ (ความเร็วและเนื้อที่ที่ต้องใช้ในการทดสอบ)

ในการทดสอบประสิทธิภาพของโครงสร้างข้อมูลแบบ ternary tree จะเป็นการ insert คำภาษาไทย จำนวน 34022 คำ ซึ่งมีความยาวในแต่ละคำไม่เกิน 40 ตัวอักษร แล้วทำการวัดความเร็วและเนื้อที่ที่ใช้ไป และทำการค้นหาคำ 1 รอบ เพื่อทำการวัดความเร็วในการค้นหา และบันทึกไว้เพื่อนำไปเปรียบเทียบ กับโครงสร้างข้อมูลแบบอื่นๆ

เนื้อที่ที่ใช้ในการเก็บข้อมูล 2,213,020 ไบต์
ความเร็วในการจัดเก็บข้อมูล 0.22 วินาที
ความเร็วในการค้นหาคำ (แบบ Successful) 0.15 วินาที
มีการประกาศใช้โหนดในการเก็บข้อมูลจำนวน 110651 โหนด

3.5.5 การทำงานของแต่ละฟังก์ชั่น

3.5.5.1 ฟังก์ชั่น -> Tptr insert(Tptr p, char *s)
พารามิเตอร์ -> 1. p เป็น pointer ของ Trie ที่จะเก็บข้อมูล
2. s เป็นสตริงที่จะเก็บใน p
จุดประสงค์ -> เพื่อเก็บสตริงใหม่ลงใน ternary tree
เรียกใช้งาน ->1. เรียกจากฟังก์ชั่น main()

2. เรียกจากตัวเอง

3.5.5.2 ฟังก์ชั่น -> void subfix(Tptr p)

พารามิเตอร์ -> p เป็น pointer ของ ternary tree

ส่งค่ากลับ -> ไม่มี

จุดประสงค์ -> เพื่อแสดงสตริงที่บรรจุอยู่ใน ternary tree

เรียกใช้งาน -> 1. จากฟังก์ชั่น main()

3.5.5.3 ฟังก์ชั่น -> Tptr walk(Tptr p,char s)

พารามิเตอร์ -> 1. p เป็น pointer ของ ternary tree

2. s เป็นตัวอักษรที่จะค้นหาใน p

ส่งค่ากลับ -> เป็น pointer ของ trie ที่มีข้อมูลตรงกับ s

จุดประสงค์ -> เพื่อหาโหนดที่เก็บตัวอักษร s ที่อยู่ในระดับความลึกเท่ากับ p

เรียกใช้งาน -> 1.จากฟังก์ชั่น find()

3.5.5.4 ฟังก์ชั่น -> Tptr find(Tptr p,char *s)

พารามิเตอร์ -> 1. p เป็น pointer ของ ternary tree

2. s เป็นสตริงที่จะค้นหาใน p

ส่งค่ากลับ -> เป็น pointer ของ trie ที่เก็บตัวอักษรสุดท้ายใน s

จุดประสงค์ -> เพื่อหาตำแหน่งของคำในในโครงสร้างข้อมูล

เรียกใช้งาน -> 1.จากฟังก์ชั่น main()

3.5.5.5 ฟังก์ชั่น -> void main()

พารามิเตอร์ -> ไม่มี

ส่งค่ากลับ -> ไม่มี

จุดประสงค์ -> 1. สร้างโครงสร้างข้อมูล

2. อ่าน input จากไฟล์และค้นหาข้อมูล

3. จับเวลาการทำงานและรายงานผล

3.5.6 สรุปข้อดีข้อเสีย

ข้อดี

1. มีความเร็วในการค้นหาข้อมูลสูง
2. ง่ายในการประยุกต์ใช้งานหรือแก้ไขเพิ่มเติม
3. ประหยัดหน่วยความจำ

4. เหมาะกับงานที่มี input ไม่แน่นอน
5. สามารถค้นหาคำทีละตัวอักษรได้

3.6 Burst Trie Data Structure

3.6.1 โครงสร้างข้อมูลในภาษาซี

```
typedef struct Bnode* Bptr;
struct Bnode
{
 char type;
};

typedef struct Dnode* Dptr;
struct Dnode
{
 char type;
 char st;
};

typedef struct Arec* ARptr;
struct Arec{
 char type;
 Bptr eq;
};

typedef struct Anode* Aptr;
struct Anode
{
 char type;
 char st;
 Arec rec[256];
};

typedef struct Tnode* Tptr;
```

```
struct Tnode
{
 char type;
 unsigned char data;
 char st;
 Tptr eq,hi,lo;
};

typedef struct Cnode* Cptr;
struct Cnode
{
 char type;
 char st;
 int num;
 Tptr eq;
};
```

เนื่องจากโครงสร้างข้อมูลแบบ burst trie มีการผสมผสายโครงสร้างข้อมูลหลายชนิดเข้าด้วยกัน จึงต้องประกาศโครงสร้างข้อมูลสำหรับโครงสร้างข้อมูลแต่ละชนิด และโครงสร้างข้อมูลพิเศษ โครงสร้างข้อมูลแบบพิเศษนี้จะถูกใช้ในลักษณะของ pointer เพื่อเป็นตัวแทนของโครงสร้างข้อมูลหลายชนิด

โครงสร้างข้อมูลธรรมดา

1. Anode เป็นโหนด ของ Access trie
2. Arec เป็น record ของ Array ใน Anode
3. Cnode เป็น Container
4. Tnode เป็น Ternary tree ที่บรรจุใน Container

โครงสร้างข้อมูลตัวแทน

1. Bnode เป็นโครงสร้างข้อมูลตัวแทนของ node ทุกชนิด
2. Dnode เป็นโครงสร้างข้อมูลตัวแทนของ Anode และ Cnode

โครงสร้างข้อมูลทุกชนิดจะมีฟิลด์ type เพื่อบ่งบองชนิดของ node โดยโครงสร้างข้อมูลแบบธรรมดาจะมีค่า type คงที่

1. Anode มีค่า type เป็น 1

2. Arec มีค่า type เป็น 2
3. Cnode มีค่า type เป็น 3
4. Tnode มีค่า type เป็น 4

แต่ในโครงสร้างข้อมูลตัวแทนจะมีค่า type ที่เป็นไปได้หลายค่า ขึ้นอยู่กับชนิดของโหนดที่ชี้อยู่

3.6.2 การค้นหาข้อมูลโครงสร้าง

โปรแกรมจะเริ่มค้นหาที่โหนด root เป็นอันดับแรก โดยการเปรียบเทียบตัวอักษรตัวแรกกับข้อมูลในโหนด root หากพบว่าไม่ตรงกันก็จะเปรียบเทียบใหม่ในโหนด next แต่ถ้าพบว่าตรงกันก็จะค้นหาตัวอักษรถัดไปจากใน child เป็นเช่นนี้ไปเรื่อยๆจนจบคำ ในการเขียนโปรแกรมได้แบ่งการทำงานออกเป็น 2 ฟังก์ชั่น โดยฟังก์ชั่น walk() ทำหน้าที่ค้นหาโหนดที่มีข้อมูลตรงกับตัวอักษร ซึ่งจะถูกเรียกใช้จากฟังก์ชั่น find() ในการค้นหาตัวอักษรตามความยาวของ String ในการค้นหาสตริง s จะใช้ code find(root,s);

```
Bptr walk(Bptr p,unsigned char s);
ARptr walk1(Aptr p,unsigned char s);
Tptr walk3(Cptr p,unsigned char s);
Tptr walk4(Tptr p,unsigned char s);

Bptr walk(Bptr p,unsigned char s)
{
  if (p) {

        switch (p->type) {

                case 1:return (Bptr)walk1((Aptr)p,s);
                case 3:return (Bptr)walk3((Cptr)p,s);
                case 4:return (Bptr)walk4((Tptr)p,s);
        }
  }
  return NULL;
}
ARptr walk1(Aptr p,unsigned char s)
```

```
{
 return &p->rec[s];
}

Tptr walk3(Cptr p,unsigned char s)
{
  return walk4(p->eq,s);
}

Tptr walk4(Tptr p,unsigned char s)
{
   if (!p) return NULL;
   if (s == p->data)
            return p;
   if (s > p->data)
            return walk4(p->hi,s);
   if (s < p->data)
            return walk4(p->lo,s);
}

Bptr find(Bptr p,unsigned char *s)
{
 Bptr o=p;
 while (*s&&o)
 {
   o=walk(o,*s);
            unsigned char *q=s;
            q++;
   if (*q&&o)
   {
           if (o->type==2) {
            o=(Bptr)((ARptr)o)->eq;
           } else {
            o=(Bptr)((Tptr)o)->eq;
```

```
        }
    }
    s++;
  }
  return o;
}
```

3.6.3 การ insert ข้อมูล

ฟังก์ชั่น insert ทำการ insert สตริงเข้าไปใน trie และจะไม่มีการ insert ถ้ามีข้อมูลอยู่แล้วจะทำการ insert สตริง s ด้วย code root=insert(root,s)

```
Bptr insert(Bptr p,unsigned char* s);
Aptr insert1(Aptr p,unsigned char* s);
Bptr insert3(Cptr p,unsigned char* s);
Tptr insert4(Tptr p,unsigned char* s);
Aptr burst(Cptr);

Bptr insert(Bptr p,unsigned char* s)
{
   if (!p)
   {
                Cptr newc=new Cnode;
                newc->type=3;
                newc->st=0;
                newc->num=1;
                newc->eq=insert4(NULL,s);
                return (Bptr)newc;
   } else {
        switch (p->type) {
                case 1:return (Bptr)insert1((Aptr)p,s);
                case 3:return (Bptr)insert3((Cptr)p,s);
                case 4:return (Bptr)insert4((Tptr)p,s);
        }
   }
```

```
    return NULL;
}

Aptr insert1(Aptr p,unsigned char* s)
{
    unsigned char ch=*s;
    ++s;
    if (*s) {
     p->rec[ch].eq=insert(p->rec[ch].eq,s);
    } else {
                    if (p->rec[ch].eq) {
                            ((Dptr)p->rec[ch].eq)->st=-1;
                    } else {
                            Cptr newc=new Cnode;
                            newc->type=3;
                            newc->st=-1;
                            newc->num=0;
                            newc->eq=NULL;
                            p->rec[ch].eq=(Bptr)newc;
                    }
    }
    return p;
}

Bptr insert3(Cptr p,unsigned char* s)
{
    ++(p->num);
            if (*s) p->eq=(Tptr)insert4(p->eq,s);
    if (p->num>limit) {return (Bptr)burst(p);}
    return (Bptr)p;
}

Tptr insert4(Tptr p,unsigned char *s)
{ if (p == NULL) {
```

```
            p = new Tnode;
            p->type=4;
            p->data = *s;
            p->st=0;
            p->lo = p->eq = p->hi = NULL;
    }
            unsigned char* q=s;
            q++;
    if (*s < p->data)
            p->lo = insert4(p->lo, s);
    else if (*s == p->data) {
            if (*q)
               {p->eq = insert4(p->eq,q);}
            else {p->st=-1;}
    } else
            p->hi = insert4(p->hi, s);
    return p;
}
```

3.6.4 ประสิทธิภาพ (ความเร็วและเนื้อที่ที่ต้องใช้ในการทดสอบ)

ในการทดสอบประสิทธิภาพของโครงสร้างข้อมูลแบบ burst trie จะเป็นการ insert คำภาษาไทย จำนวน 34022 คำ ซึ่งมีความยาวในแต่ละคำไม่เกิน 40 ตัวอักษร แล้วทำการวัดความเร็วและเนื้อที่ที่ใช้ไป และทำการค้นหาคำ 1 รอบ เพื่อทำการวัดความเร็วในการค้นหา และบันทึกไว้เพื่อนำไปเปรียบเทียบ กับโครงสร้างข้อมูลแบบอื่นๆ

Limit 16

เนื้อที่ที่ใช้ในการเก็บข้อมูล 3,552,372 ไบต์

ความเร็วในการจัดเก็บข้อมูล 0.11 วินาที

ความเร็วในการค้นหาคำ (แบบ Successful) 0.12 วินาที

มีการประกาศใช้ Access trie โหนดในการเก็บข้อมูลจำนวน 895 โหนด

มีการประกาศใช้ Container ในการเก็บข้อมูลจำนวน 10078 Container

มีการประกาศใช้ Ternary tree โหนดในการเก็บข้อมูลจำนวน 99681 โหนด

Limit 32

เนื้อที่ที่ใช้ในการเก็บข้อมูล 2,633,620 ไบต์
ความเร็วในการจัดเก็บข้อมูล 0.12 วินาที
ความเร็วในการค้นหาคำ (แบบ Successful) 0.12 วินาที
มีการประกาศใช้ Access trie โหนดในการเก็บข้อมูลจำนวน 437 โหนด
มีการประกาศใช้ Container ในการเก็บข้อมูลจำนวน 6644 Container
มีการประกาศใช้ Ternary tree โหนดในการเก็บข้อมูลจำนวน 103573 โหนด

Limit 64
เนื้อที่ที่ใช้ในการเก็บข้อมูล 2,197,620 ไบต์
ความเร็วในการจัดเก็บข้อมูล 0.11 วินาที
ความเร็วในการค้นหาคำ (แบบ Successful) 0.12 วินาที
มีการประกาศใช้ Access trie โหนดในการเก็บข้อมูลจำนวน 218 โหนด
มีการประกาศใช้ Container ในการเก็บข้อมูลจำนวน 4173 Container
มีการประกาศใช้ Ternary tree โหนดในการเก็บข้อมูลจำนวน 106263 โหนด

Limit 128
เนื้อที่ที่ใช้ในการเก็บข้อมูล 1,970,648 ไบต์
ความเร็วในการจัดเก็บข้อมูล 0.13 วินาที
ความเร็วในการค้นหาคำ (แบบ Successful) 0.12 วินาที
มีการประกาศใช้ Access trie โหนดในการเก็บข้อมูลจำนวน 103 โหนด
มีการประกาศใช้ Container ในการเก็บข้อมูลจำนวน 2381 Container
มีการประกาศใช้ Ternary tree โหนดในการเก็บข้อมูลจำนวน 108170 โหนด

Limit 1024
เนื้อที่ที่ใช้ในการเก็บข้อมูล 1,797,044 ไบต์
ความเร็วในการจัดเก็บข้อมูล 0.15 วินาที
ความเร็วในการค้นหาคำ (แบบ Successful) 0.13 วินาที
มีการประกาศใช้ Access trie โหนดในการเก็บข้อมูลจำนวน 14 โหนด
มีการประกาศใช้ Container ในการเก็บข้อมูลจำนวน 481 Container
มีการประกาศใช้ Ternary tree โหนดในการเก็บข้อมูลจำนวน 110159 โหนด

3.6.5 การทำงานของแต่ละฟังก์ชั่น

3.6.5.1 ฟังก์ชั่น -> Bptr insert(Bptr p,unsigned char* s);

พารามิเตอร์ -> 1. p เป็น pointer ของ Trie ที่จะเก็บข้อมูล

2. s เป็นสตริงที่จะเก็บใน p

จุดประสงค์ -> เพื่อเก็บสตริงใหม่ลงใน trie

เรียกใช้งาน ->1. เรียกจากฟังก์ชั่น main()

2. เรียกจากฟังก์ชั่น insert1()

3.6.5.2 ฟังก์ชั่น ->Aptr insert1(Aptr p,unsigned char* s);

พารามิเตอร์ -> 1. p เป็น pointer ของ Trie ที่จะเก็บข้อมูล

2. s เป็นสตริงที่จะเก็บใน p

จุดประสงค์ -> เพื่อเก็บสตริงใหม่ลงใน trie สำหรับกรณีที่ p เป็น Access trie

เรียกใช้งาน -> เรียกจากฟังก์ชั่น insert()

3.6.5.3 ฟังก์ชั่น -> Bptr insert3(Cptr p,unsigned char* s);

พารามิเตอร์ -> 1. p เป็น pointer ของ Trie ที่จะเก็บข้อมูล

2. s เป็นสตริงที่จะเก็บใน p

จุดประสงค์ -> เพื่อเก็บสตริงใหม่ลงใน trie สำหรับกรณีที่ p เป็น Container

เรียกใช้งาน -> เรียกจากฟังก์ชั่น insert()

3.6.5.4 ฟังก์ชั่น ->Tptr insert4(Tptr p,unsigned char* s);

พารามิเตอร์ -> 1. p เป็น pointer ของ Trie ที่จะเก็บข้อมูล

2. s เป็นสตริงที่จะเก็บใน p

จุดประสงค์ -> เพื่อเก็บสตริงใหม่ลงใน trie สำหรับกรณีที่ p เป็น Ternary tree

เรียกใช้งาน ->1. เรียกจากฟังก์ชั่น insert()

2. เรียกจากฟังก์ชั่น insert3()

3. เรียกจากตัวเอง

3.6.5.5 ฟังก์ชั่น -> Aptr burst(Cptr);

พารามิเตอร์ -> p เป็น pointer ของ trie

ส่งค่ากลับ -> เป็น pointer ของโหนดใหม่ที่เกิดจากการ burst

จุดประสงค์ -> เพื่อ burst Container ที่มีจำนวน record เกิน limit เป็น Anode

เรียกใช้งาน -> เรียกจากฟังก์ชั่น insert3()

3.6.5.6 ฟังก์ชั่น -> void transfer(Tptr p,Aptr n)

พารามิเตอร์ -> 1. p เป็น pointer ของ trie

2. n เป็น Anode

ส่งค่ากลับ -> ไม่มี

จุดประสงค์ -> เพื่อย้ายข้อมูลจาก Ternary tree ไปยัง Anode

เรียกใช้งาน -> 1. เรียกจากฟังก์ชั่น burst()

3.6.5.7 ฟังก์ชั่น -> Bptr walk(Bptr p,unsigned char s);

พารามิเตอร์ -> 1. p เป็น pointer ของ Trie

2. s เป็นตัวอักษรที่จะค้นหาใน p

ส่งค่ากลับ -> เป็น pointer ของ trie ที่มีข้อมูลตรงกับ s

จุดประสงค์ -> เพื่อหาโหนดที่เก็บตัวอักษร s ที่อยู่ในระดับความลึกเท่ากับ p

เรียกใช้งาน -> 1.เรียกจากฟังก์ชั่น find()

3.6.5.8 ฟังก์ชั่น -> ARptr walk1(Aptr p,unsigned char s);

พารามิเตอร์ -> 1. p เป็น pointer ของ Trie

2. s เป็นตัวอักษรที่จะค้นหาใน p

ส่งค่ากลับ -> เป็น pointer ของ trie ที่มีข้อมูลตรงกับ s

จุดประสงค์ -> เพื่อหาโหนดที่เก็บตัวอักษร s ที่อยู่ในระดับความลึกเท่ากับ p สำหรับกรณี p เป็น Access trie

เรียกใช้งาน ->เรียกจากฟังก์ชั่น walk()

3.6.5.9 ฟังก์ชั่น -> Tptr walk3(Cptr p,unsigned char s);

พารามิเตอร์ -> 1. p เป็น pointer ของ Trie

2. s เป็นตัวอักษรที่จะค้นหาใน p

ส่งค่ากลับ -> เป็น pointer ของ trie ที่มีข้อมูลตรงกับ s

จุดประสงค์ -> เพื่อหาโหนดที่เก็บตัวอักษร s ที่อยู่ในระดับความลึกแรกของContainer p

เรียกใช้งาน -> เรียกจากฟังก์ชั่น walk()

3.6.5.10 ฟังก์ชั่น -> Bptr Tptr walk4(Tptr p,unsigned char s);

พารามิเตอร์ -> 1. p เป็น pointer ของ Trie

2. s เป็นตัวอักษรที่จะค้นหาใน p

ส่งค่ากลับ -> เป็น pointer ของ trie ที่มีข้อมูลตรงกับ s

จุดประสงค์ -> เพื่อหาโหนดที่เก็บตัวอักษร s ที่อยู่ในระดับความลึกเท่ากับ p

เรียกใช้งาน -> 1. เรียกจากฟังก์ชั่น walk()

2. เรียกจากฟังก์ชั่น walk3()

3. เรียกจากตัวเอง

3.6.5.11 ฟังก์ชั่น -> Bptr find(Bptr p,unsigned char *s)

พารามิเตอร์ -> 1. p เป็น pointer ของ Trie

2. s เป็นสตริงที่จะค้นหาใน p

ส่งค่ากลับ -> เป็น pointer ของ trie ที่เก็บตัวอักษรสุดท้ายใน s

จุดประสงค์ -> เพื่อหาตำแหน่งของคำในในโครงสร้างข้อมูล

เรียกใช้งาน -> 1.จากฟังก์ชั่น main()

3.6.5.12 ฟังก์ชั่น -> void main()

พารามิเตอร์ -> ไม่มี

ส่งค่ากลับ -> ไม่มี

จุดประสงค์ -> 1. สร้างโครงสร้างข้อมูล

2. อ่าน input จากไฟล์และค้นหาข้อมูล

3. จับเวลาการทำงานและรายงานผล

3.6.6 สรุปข้อดีข้อเสีย

ข้อดี

1. มีความเร็วในการค้นหาข้อมูลสูงมาก
2. เหมาะกับงานที่มี input ไม่แน่นอน
3. สามารถค้นหาคำทีละตัวอักษรได้

ข้อเสีย

1. มีโครงสร้างข้อมูลซับซ้อน ยากต่อการนำไปพัฒนาต่อ
2. ใช้เนื้อที่หน่วยความจำมาก

บทที่ 4
เทคนิคการตัดคำ

4.1 การตัดคำแบบ 1 parse

ในการตัดคำแบบ 1 parse จะใช้เทคนิคการแบ่ง token โดย 1 token หมายถึง 1 กรณีที่เป็นไปได้ในการแบ่งคำ 1 ประโยค

ภายใน Token ประกอบด้วย

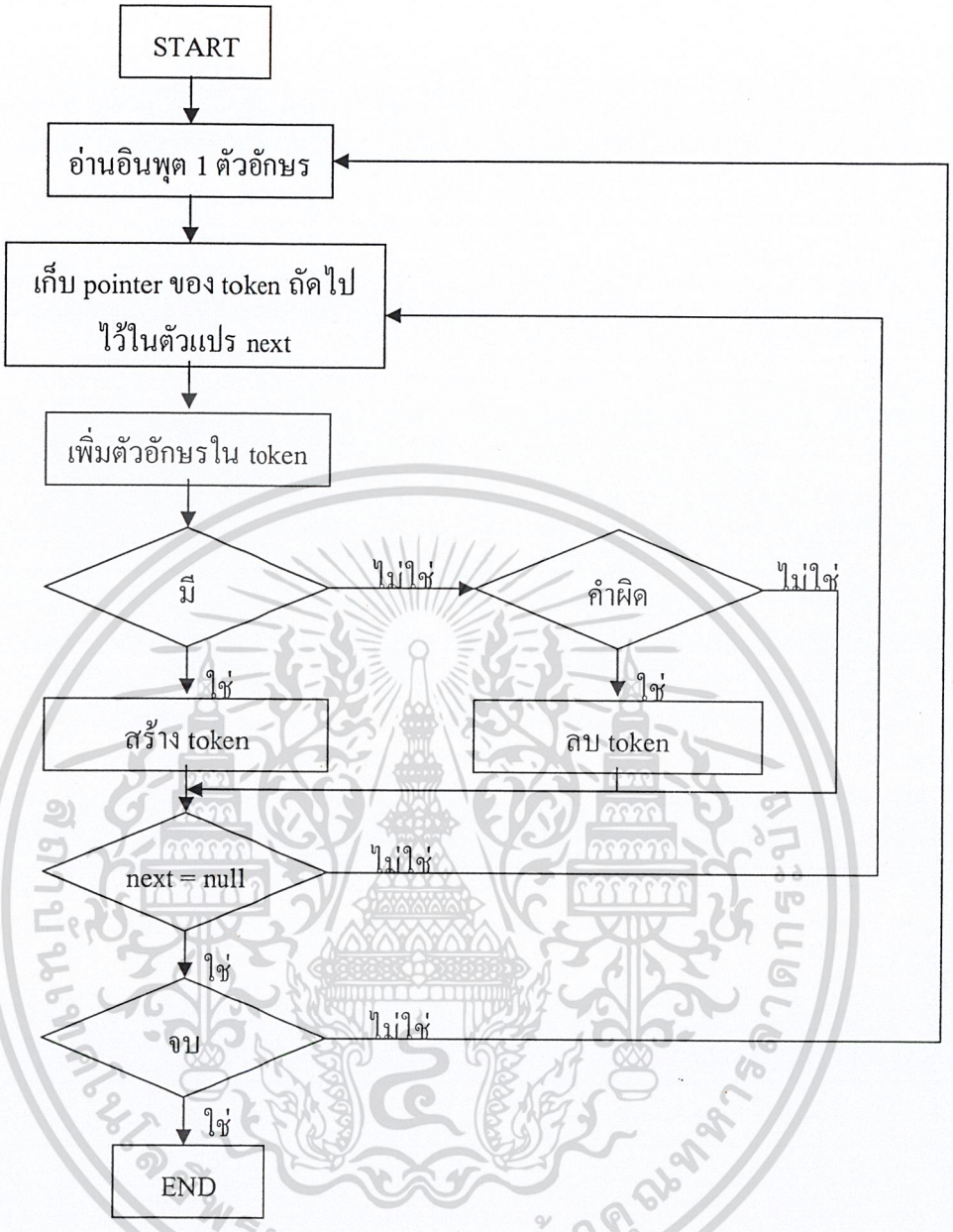
1. วิธีการแบ่งคำ
2. ตำแหน่งของ node ในพจนานุกรมที่ Token กำลังอ่านอยู่
3. สถานะของ Token
4. ตัวชี้ไปยัง Token ถัดไป

หากมีวิธีการตัดคำที่เป็นไปได้หลายวิธี จะมี Token หลาย Token ต่อกันในลักษณะ link list

เริ่มต้น ก่อนการรับ input จะสร้าง Token ขึ้นมา 1 Token โดยมีตำแหน่งของ node ในพจนานุกรมอยู่ที่ root แล้วจึงรับ input เข้ามาทีละ 1 ตัวอักษร แล้วนำไปประมวลผล โดยใช้ input เป็น key ในการค้นหาคำใน พจนานุกรม หาก input ที่รับเข้ามาทำให้เกิดเป็นคำ สถานะของ node ในพจนานุกรมที่ Token กำลังอ่านอยู่จะเป็น -1 จะต้องมีการแบ่ง Token ออกเป็น 2 Token โดย Token แรกจะคงเหมือนเดิม แต่ Token ที่สองจะตัดคำออกเป็นคำใหม่ แล้วย้ายตำแหน่งของ node ในพจนานุกรมไปที่ root เพื่อเริ่มต้นอ่านคำใหม่ แต่ถ้า input ที่รับเข้ามาทำให้เกิดเป็นคำที่ไม่มีความหมาย node ในพจนานุกรมที่ Token กำลังอ่านอยู่จะเป็น null แสดงว่าวิธีการตัดคำของ Token นั้นผิด จะต้องลบ Token นั้นทิ้ง แล้วคืนหน่วยความจำให้ระบบ

หลังจากประมวลผล Token เสร็จแล้ว ก็จะประมวลผล Token อื่นๆใน list ด้วย input เดียวกันจนจบ link list แล้วจึงเริ่มอ่าน input ใหม่

เมื่อประมวลผลจนจบ input ก็จะเกิด Token มากมาย Token ที่มีตำแหน่งของ node ในพจนานุกรมอยู่ที่ root แสดงว่าสามารถอ่านได้จบคำพอดี จะเป็นวิธีการตัดคำที่ถูกต้อง



รูปที่ 4-1 แสดงลำดับการทำงานของ 1 parse

4.2 การกำจัด Token ที่ไม่จำเป็น แบบจำกัดจำนวน Token

ในกรณีที่เป็นคำประสมหรือคำใดๆที่สามารถแบ่งได้เป็นคำที่มีความหมาย 2 คำขึ้นไป เช่น นาที โทรคมนาคม โทรศัพท์ หน้าจอ จานบิน ฯลฯ จะทำให้เกิด Token ใหม่เป็นจำนวนเท่าตัว โดยที่ไม่มีการถูกลบทิ้ง เพราะมีตำแหน่งจบคำที่เดียวกัน ทำให้ใช้หน่วยความจำปริมาณมากเกินไป เช่น หากมี คำประสมใน input จำนวน 30 คำ จะทำให้เกิด Token อย่างน้อย 2^{30} = 1,073,741,824 Token จึงไม่สามารถนำมาใช้ได้ในทางปฏิบัติ

ในความเป็นจริงแล้ว Output ที่ต้องการไม่ใช่ทุกกรณี แต่ต้องการคำที่ยาวที่สุดที่เป็นไปได้ เช่นคำว่า โทรคมนาคม สามารถแบ่งได้เป็น โทรคมนาคม, โทรคม นา คม, โทร คมนาคม, โทร คม นา คม แต่ต้องการ output ที่เป็น โทรคมนาคม เท่านั้น จึงต้องมีการตัด Token ที่ไม่จำเป็นทิ้งไป ซึ่งแบบแรกเป็นวิธีการจำกัดจำนวน Token หาก Token มีจำนวนเกินกว่าที่กำหนดไว้ ก็จะตัวส่วนที่เกินทิ้งไป และการสร้าง Token ใหม่จะนำ Token ที่สร้างใหม่ต่อท้าย Token เดิมเสมอ ทำให้ส่วนที่เหลืออยู่เป็นวิธีการแบ่งคำที่ไม่แบ่งย่อยจนเกินไป การกำจัด Token แบบนี้มีข้อดีคือ สามารถจำกัดขนาดการใช้หน่วยความจำและความเร็วในการทำงานของโปรแกรมได้ แต่ก็มีข้อเสียคือ ไม่สามารถรับประกันได้ว่าจะไม่ได้ตัด Token ที่มีการแบ่งคำอย่างถูกต้องทิ้งไป

4.3 การกำจัด Token ที่ไม่จำเป็น แบบตรวจสอบตำแหน่ง

การกำจัด Token ที่ไม่จำเป็น แบบตรวจสอบตำแหน่ง เป็นเทคนิคการกำจัด Token อีกแบบหนึ่ง โดยยึดหลักว่า Token 2 Token ขึ้นไปที่มีตำแหน่งจบคำที่เดียวกัน จะมีการประมวลผลในตัวอักษรต่อไปในลักษณะเดียวกัน จึงให้เก็บเฉพาะ Token ที่มีการตัดคำที่ตำแหน่งนี้เป็น Token แรกเท่านั้น ซึ่งก็คือ Token ที่มีการตัดคำที่มีการแบ่งคำน้อย (คำยาว) เพราะอยู่ส่วนบนของ list

การตัดคำโดยใช้วิธีการกำจัด Token แบบตรวจสอบตำแหน่ง ทำให้ Token ที่เหลืออยู่มีน้อยมากๆ จึงมีความเร็วมากกว่าและใช้หน่วยความจำน้อยกว่าวิธีการกำจัด Token แบบจำกัดจำนวน Token โดยที่ไม่ต้องกลัวว่าจะตัด Token ที่ถูกต้องทิ้งไป

ในการกำจัด Token ที่ไม่จำเป็น ทั้ง 2 แบบ ทำให้โปรแกรม ทำงานได้เร็ว และมีประสิทธิภาพ แต่มีข้อเสียคือทำให้ไม่สามารถ Output กรณีที่เป็นไปได้ครบทุกกรณี

4.4 การตรวจสอบคำผิด

ในการตัดคำ หาก input มีคำผิดอยู่แม้เพียง 1 ตัวอักษร จะทำให้ไม่สามารถตัดคำได้ เพราะ Token ทั้งหมดจะถูกลบทิ้ง ในหัวข้อนี้จะอธิบายวิธีการจัดการกับ input ที่มีความผิดพลาด โดยสามารถกำหนดช่วงของคำผิดและสามารถตัดคำในส่วนที่เหลือได้อย่างปกติ

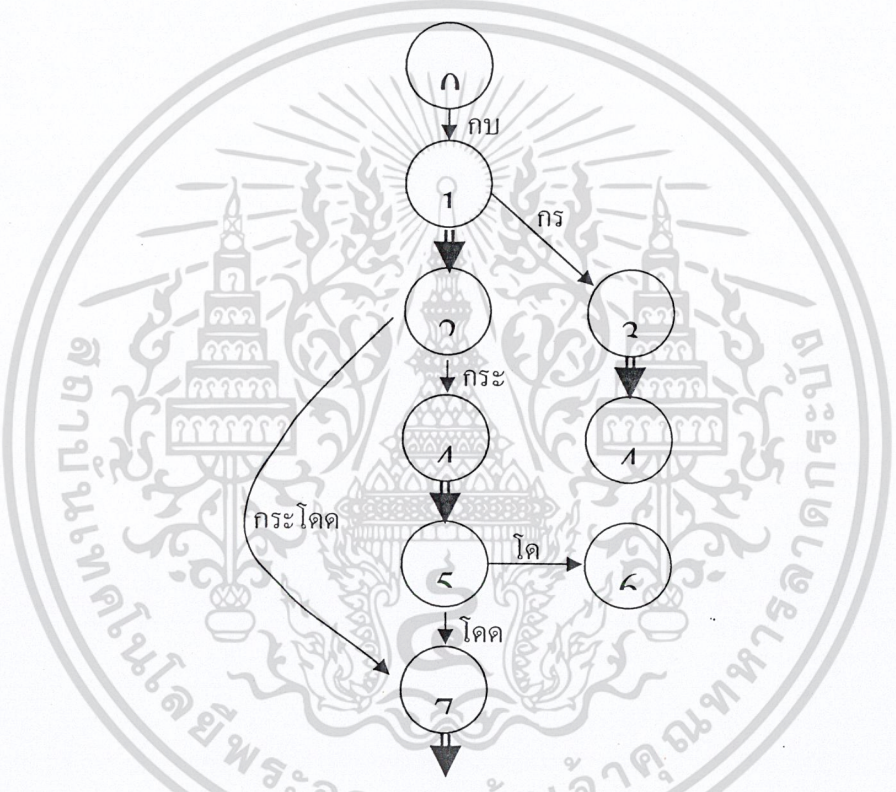
ในการประมวลผลคำในแต่ละรอบ หากมีการลบ Token ออกจาก list Token แรกที่ถูกลบจะถูกลบออกจาก list เท่านั้น ไม่ได้ลบออกจากหน่วยความจำและจะเก็บ Token นั้นไว้ในฐานะ Token ที่ไปได้ไกลที่สุด จนกว่าจะประมวลผลเสร็จครบทุก Token

เมื่อประมวลผลเสร็จครบทุก Token แล้ว หากทุก Token ถูกลบหมดจนไม่มี Token เหลืออยู่ใน list เลย แสดงว่ามีคำผิดอยู่ใน input จะต้องนำ Token ที่ไปได้ไกลที่สุดมาจำกัดขอบเขตของคำผิดและประมวลผลต่อ ซึ่งทำได้โดยนำ Token มาสร้างเป็น Token ใหม่ จำนวนเท่ากับความยาวของคำที่ยาวที่สุด

แล้วเพิ่มคำลงไปใน Token 1 คำ โดยคำนี้คือคำผิด และกำหนดความยาวของคำนี้ในแต่ละ Token ต่างๆกัน โดยคำผิดที่มีความยาวน้อยจะอยู่ส่วนบนของ list จากนั้นจึงนำ Token เหล่านี้มาประมวลผลต่อในตัวอักษรถัดไปตามปกติ Token ที่กำหนดช่วงของคำผิดไม่ถูกต้องจะถูกลบไปเองโดยอัตโนมัติ คงเหลือแต่ Token ที่ดีที่สุดเท่านั้น

4.5 การตัดคำแบบ 2 parse

ในการตัดคำแบบ 2 parse เป็นการตัดคำโดยมีการทำงาน 2 รอบ โดยจะรับ input ทีละตัวอักษร แล้วทำการตัดคำไปเก็บไว้ในตารางคำก่อน แล้วจึงนำคำในตารางคำมารวมกลับเป็นประโยคที่สมบูรณ์



รูปที่ 4-2 แสดงลักษณะการทำงานของ 2 parse

4.6 การตัดคำไปเก็บในตารางคำ

ในการตัดคำ จะใช้เทคนิคการแบ่ง token โดย 1 token หมายถึงคำ 1 คำที่ยังไม่สมบูรณ์ในการตัดคำนั้น จะต้องมีการตรวจสอบคำในพจนานุกรม ซึ่งจะต้องใช้ พจนานุกรมที่สามารถรับ input ทีละตัวอักษรได้ เมื่อกระบวนการตัดคำอ่าน input เข้ามา 1 ตัวอักษร ก็จะส่งไปให้พจนานุกรมตรวจสอบ หากเกิดเป็นคำที่มีความหมาย ก็จะนำคำไปเก็บไว้ในตารางคำ และสร้าง token ใหม่เพื่อรอรับตัวอักษรแรกของคำต่อไป ส่วน token เดิมนั้น จะถูกเก็บไว้เพื่อตรวจหาคำที่มีความยาวกว่า และจะถูกทำลายเมื่อ token นั้นกลายเป็นคำผิด

ตำแหน่งเริ่มต้น	ตำแหน่งจบ (คำที่บันทึก)
0 (“ก”)	1 (“กบ”)
1 (“บ”)	
2 (“ก”)	7 (“กระโดด”), 4 (“กระ”), 3 (“กร”)
3 (“ร”)	
4 (“ะ”)	
5 (“โ”)	7 (“โดด”), 6 (“โด”)
6 (“ด”)	
7 (“ด”)	

ตารางที่ 4-1 แสดงตารางการตัดคำ

4.7 ตารางคำ

ตารางคำมีลักษณะเป็น Array ของ link list โดย ดัชนีของ Array คือตำแหน่งเริ่มต้นของคำ และมี pointer ชี้ไปยังจุดสิ้นสุดของคำ ซึ่งเก็บในลักษณะของ link list ทำให้สามารถมีคำที่เริ่มต้นที่ตำแหน่งเดียวกันได้หลายคำ

4.8 การนำคำมารวมเป็นประโยคที่สมบูรณ์

ในการรวมคำเป็นประโยค จะใช้วิธีการแตก token โดย 1 token หมายถึง 1 วิธีในการรวมคำ ในการเพิ่มคำลงใน token นั้น หากคำที่เก็บใน token สิ้นสุดที่ n จะเพิ่มคำโดยหาคำจากตารางคำที่มีจุดเริ่มต้นที่ n+1 หากมีมากกว่า 1 คำ ก็จะทำการแตก token เพื่อเก็บเส้นทางทุกเส้นทางที่เป็นไปได้ เมื่อคำที่เก็บใน token สิ้นสุดที่ตำแหน่งสุดท้ายของ ประโยค ก็ถือว่าวิธีการรวมคำเสร็จสิ้นแล้ว 1 วิธี

4.9 การกำจัดกรณีที่ไม่จำเป็น

กรณีที่ไม่จำเป็นคือกรณีที่เกิดจากคำประสม เช่นคำว่า โทรคมนาคม สามารถแบ่งได้เป็น

โทรคมนาคม

โทรคม นา คม

โทร คมนาคม

โทร คม นา คม

ซึ่งเราต้องการเพียงกรณีที่ยาวที่สุดเท่านั้น คือ โทรคมนาคม แต่ในบางกรณีก็สามารถแบ่งคำได้เป็น 2 แบบ เช่นคำว่า กรมดอก สามารถแบ่งได้เป็น

กรม ดอก

กร มด อก

ซึ่งเราต้องการเก็บไว้ทั้ง 2 กรณี ดังนั้น ในการกำจัดกรณีที่ไม่จำเป็น เราเพียงแต่เพิ่มชุดคำสั่งแทรกลงไประหว่างการทำงานในรอบแรก และการทำงานในรอบที่สอง โดยชุดคำสั่งนี้จะตรวจสอบตารางคำ หากมีคำ 2 คำ เริ่มต้นที่ตำแหน่งเดียวกัน จะต้องทดลองค้นหาเส้นทางโดยใช้คำที่สั้นกว่าเป็นจุดเริ่มต้นแล้วค้นหาต่อไปเรื่อยๆในคำถัดไป หากไปสิ้นสุดที่ตำแหน่งเดียวกับคำที่ยาวกว่า แสดงว่าเป็นกรณีที่ต้องตัดทิ้งเช่น

(คำยาว) โทรคมนาคม

(คำสั้น) โทรคม ---ค้นหา---> นา ---ค้นหา---> คม

แต่ถ้าไปสิ้นสุดที่ตำแหน่งไกลกว่าคำที่ยาวกว่า แสดงว่าคำๆนั้นสามารถแบ่งได้ 2 แบบ เช่น

(คำยาว) กรม

(คำสั้น) กร ---ค้นหา---> มด

หลังจากจบการทำงานของชุดคำสั่งนี้แล้ว ก็จะทำงานในรอบที่ 2 ตามปกติ คำตอบที่ได้จะเป็นทุกกรณีที่ไม่มีการแบ่งคำประสม

4.10 การตรวจสอบคำผิด

4.10.1 การทำงานในรอบแรก

หากพบคำผิด token ทั้งหมดจะถูกลบ และสร้าง token ว่างขึ้นมาใหม่โดยเริ่มค้นหาที่ตัวอักษรถัดไป

4.10.2 ลักษณะของข้อมูลในตารางคำ

หากมีคำผิด ข้อมูลในตารางคำจะไม่เชื่อมต่อกัน เช่น หากคำผิดอยู่ ณ ตำแหน่งที่ (4,4) เส้นทางที่ไปได้ไกลที่สุดจะเริ่มต้นจากตำแหน่งที่ 0 และสิ้นสุดที่ตำแหน่งที่ 3 และจะมีเส้นทางจากตำแหน่งที่ 5 ไปจนจบประโยค

4.10.3 การทำงานในรอบที่ 2

หากมีคำผิด token ทั้งหมดจะถูกลบ จะนำเส้นทางที่ไกลที่สุดมาคำนวณต่อ โดยเริ่มจากตำแหน่งที่มีจุดเริ่มต้นของคำถัดไป โดยดูจากตารางคำ

4.11 สรุป

การตัดคำทั้งแบบ 1 parse และ 2 parse เป็นการตัดคำที่ทำงานได้อย่างรวดเร็ว แต่การตัดคำแบบ 1 parse มีข้อดีคือ สามารถตัดคำได้แม้ว่ายังอ่าน input ไม่จบจึงเหมาะกับงานที่มี input เข้ามาเรื่อยๆ แต่ก็มีข้อเสียคือ ไม่สามารถให้คำตอบได้ทุกกรณีซึ่งในการตัดคำแบบ 2 parse สามารถให้คำตอบได้ทุกกรณี แต่จำเป็นต้องอ่าน input ให้จบเสียก่อนจึงจะเริ่มทำการตัดคำได้

บทที่ 5
หลักภาษาไทย

อักษรไทยปัจจุบันแบ่งออกเป็น 4 ชนิด คือ สระ, พยัญชนะ, วรรณยุกต์และเลข โดยอักษรทั้ง 4 ชนิดนี้ ตั้งขึ้นสำหรับใช้แทนเสียง หรือจารึกเสียง เสียงในภาษาไทยจำแนกออกได้เป็น 3 ชนิด คือ เสียงแท้ได้แก่เสียงสระ เสียงแปรได้แก่เสียงพยัญชนะ และเสียงผันหรือเสียงดนตรีซึ่งก็คือเสียงที่มีจังหวะน้ำหนัก และทำนองสูง ๆ ต่ำๆ เป็นทำนองเสียงที่ผันขึ้นลงให้มีเสียงเสนาะ และจำแนกเสียงให้ผิดแผกแตกต่างออกไปเพื่อกำหนดความหมายให้แม่นยำขึ้น

เสียงผันหรือเสียงดนตรี จะรู้ได้จากรูปวรรณยุกต์ที่กำกับอยู่ข้างบนตัวอักษร แต่ถ้าไม่มีรูปวรรณยุกต์กำกับอยู่ ต้องสังเกตระดับเสียงของคำที่เปล่งออกมา วิธีสังเกตเสียงวรรณยุกต์นี้ จะได้อธิบายไว้ในตอนที่ว่าด้วยวิธีผันตัวอักษร 3 หมู่ต่อไป แต่ความจริงเสียงผันหรือเสียงดนตรี นอกจากใช้วรรณยุกต์แทนแล้ว ยังต้องอาศัย "ไตรยางค์" ด้วย

5.1 สระ

สระแปลว่าออกเสียงได้ตามลำพัง, ในภาษาพูดหมายถึงกระแสเสียงที่มีทำนองสูงต่ำซึ่งเปล่งออกมาตามจังหวะนิยม, ในภาษาหนังสือหมายถึงเครื่องหมายใช้แทนเสียงที่เปล่งออกมาเอง โดยสระ 32 ตัวเกิดจากรูปต่างๆ ไม่ซ้ำกัน 21 รูปดังนี้

ะ – วิสรรชนีย์	ั – หันอากาศ หรือ ไม้ผัด
็ – ไม้ไต่คู้ หรือไม้ตายคู้	า – ลากข้าง
ิ – พินทุ หรือ พินทุอิ	่ – ฝนทอง หรือ ฟันหนูหรือ มูสิกทันต์
์ – หยาดน้ำค้าง หรือ นิคหิต	ุ – ตีนเหยียด หรือ ลากตีน
ู – ตีนคู้ หรือ ลากตีน	เ – ไม้หน้า
ใ – ไม้ม้วน	ไ – ไม้มลาย
โ – ไม้โอ	ฤ – รึ
ฤา – รือ	ฦ – ลึ
ฦา – ลือ	ย – ตัวยอ
รร – ร หัน	ว – ตัววอ
อ – ตัวออ	

เมื่อนำสระรูปต่างๆ มาประสมกันจะเกิดเป็นเสียงสระที่สามารถนำมาใช้ได้ทั้งหมด 32 เสียงด้วยกันซึ่งสระทั้ง 32 เสียงนั้นมีดังนี้

อะ	อา	อิ	อี	อึ	อื	อุ	อู	เอะ	เอ
แอะ	แอ	โอะ	โอ	เอาะ	ออ	เออะ	เออ	เอียะ	เอีย
เอือะ	เอือ	อัวะ	อัว	ฤ	ฤา	ฦ	ฦา	อำ	ใอ
ไอ	เอา								

ซึ่งสระ 21 รูปนี้มีวิธีการเขียนรูปสระ สรุปได้ดังนี้

1. เขียนไว้หน้าพยัญชนะมี 4 รูป : เ ใ ไ โ
2. เขียนไว้หลังพยัญชนะมี 7 รูป : ะ า ฤ ย ร ว อ
3. เขียนไว้บนพยัญชนะมี 5 รูป : ั ็ ิ ์ ำ
4. เขียนไว้ล่างพยัญชนะมี 2 รูป : ุ ู
5. เขียนโดดๆ ไม่ต้องประสมกับพยัญชนะมี 4 รูป : ฤ ฤา ฦ ฦา
6. ใช้ได้ตามลำพังโดยไม่ต้องประสมกับสระรูปอื่นมี 17 รูป : ะ ั า ็ ิ ุ ู เ ใ ไ โ ฤ ฤา ฦ ฦา ร อ
7. ต้องประสมกับสระรูปอื่นมี 4 รูป : ่ ำ ย ว
8. ใช้ตามลำพังก็ได้ ใช้ควบกับพยัญชนะก็ได้ ออกเสียงคล้ายมี ร อยู่ด้วยมี 1 รูป : ฤ
9. ใช้เป็นสระก็ได้ ใช้เป็นพยัญชนะก็ได้มี 4 รูป : ย ร ว อ

สระ 21 รูปนี้เมื่อนำไปประสมกันเข้า จะเกิดเป็นสระต่างๆ ถึง 32 ตัว มีเสียง 32 เสียง ซึ่งนับว่าพอใช้ในภาษาไทย และเพราะเหตุที่เรามีสระมากเช่นนี้ เราจึงสามารถเขียนคำที่มาจากภาษาอื่นได้ถูกต้องหรือใกล้ชิดกับสำเนียงของเขาที่สุด นอกจากนั้นเรายังมีรูปวรรณยุกต์บังคับให้เสียงชัดเจนแน่นอนยิ่งขึ้นอีกด้วย จึงนับว่าภาษาไทยเป็นภาษาที่มีสำเนียงสมบูรณ์ภาษาหนึ่ง

5.2 เสียงสระ

สระ 32 ตัวจำแนกออกเป็นสระเสียงสั้น เรียกชื่อว่า รัสสระ , เสียงยาว เรียกว่า ทีฆสระ , เสียงเบา เรียกว่า ลหุ , และเสียงหนัก เรียกว่า ครุ โดยมีตารางเทียบเสียงดังนี้

เสียงสระ			
สั้น 18 (รัสสระ)	ยาว 14 (ทีฆสระ)	เบา 14 (ลหุ)	หนัก 18 (ครุ)
อะ	อา	อะ	อา

อึ	อึ	อึ	อึ
อุ	อู	อุ	อู
เอะ	เอ	เอะ	เอ
แอะ	แอ	แอะ	แอ
โอะ	โอ	โอะ	โอ
เอาะ	ออ	เอาะ	ออ
เออะ	เออ	เออะ	เออ
เอียะ	เอีย	เอียะ	เอีย
เอือะ	เอือ	เอือะ	เอือ
อัวะ	อัว	อัวะ	อัว
ฤ	ฤา	ฤ	ฤา
ฦ	ฦา	ฦ	ฦา
อำ			อำ
ใอ			ใอ
ไอ			ไอ
เอา			เอา
			สระสั้น + ตัวสะกด

ตาราง 5-1 ตารางเปรียบเทียบเสียง

5.3 การใช้สระ

สระต่างๆ ที่เราเปล่งสำเนียงออกมาทางภาษาพูด ย่อมไม่มีพลิกแพลงอย่างไร เพียงแต่ลั่นสำเนียงให้ชัดเจนเป็นที่หมายรู้กันได้ก็นับว่าพอแก่ความต้องการในทางภาษาแล้ว แต่ในภาษาหนังสือ ต้องการแสดงความหมายประจักษ์ทางตาเป็นสำคัญ เพราะฉะนั้นจึงมีระเบียบและวิธีเขียนที่ต้องอาศัยหลักเกณฑ์ละเอียดประณีตกว่าภาษาพูด เพราะนอกจากจะเป็นเครื่องหมายให้รู้ทางตาแล้ว ยังต้องเป็นเครื่องหมายนำในการออกเสียงให้ถูกต้องชัดเจนด้วย ผู้ศึกษาจึงจำเป็นต้องรู้หลักการใช้และการเขียนให้เหมาะกับยุคสมัยที่ภาษาได้เจริญก้าวหน้าไปตามจังหวะของมัน เพราะฉะนั้นการใช้และการเขียนจึงมักมีการเปลี่ยนแปรแก้ไขได้เป็นครั้งคราว ต่อไปจะได้อธิบายหลักเกณฑ์การใช้สระบางตัวที่มีเสียงซ้ำกันแต่มีรูปต่างกันหรือมีวิธีเขียนที่จำต้องกำหนดเป็นพิเศษ ส่วนสระที่เขียนตามปกติ ไม่มีวิธีพลิกแพลงอย่างไร จะไม่อธิบายไว้

5.3.1 สระอะ มีวิธีใช้ 4 อย่างคือ

- **คงรูป** คือ ต้องประวิสรรชะนีย์ (ะ) ข้างหลังพยางค์ทุกพยางค์ที่ออกเสียง อะ ในกรณีดังนี้ คือ เป็นพยางค์ของคำไทยแท้ เช่น กะบะ กะทะ มะระ ปะทะ ฯลฯ มีข้อยกเว้นบ้างซึ่งจะกล่าวในหัวข้อถัดไป, เป็นพยางค์สุดท้ายของคำบาลี สันสกฤต

ข้อยกเว้นบ้างซึ่งจะกล่าวในหัวข้อถัดไป, เป็นพยางค์สุดท้ายของคำบาลี สันสกฤต และภาษาอื่นๆ เช่น สรณะ คณะ อิสระ อาสนะ เอเดนเบอระ ฯลฯ, เป็นพยางค์ของคำในภาษาอื่นนอกจากบาลีสันสกฤตและภาษาตระกูลอินเดีย-ยุโรป เช่น บะหมี่ แป๊ะซะ ตือบะ ซากุระ โอสุกะ มะเดหวี มะงุมมะงาหลา ฯลฯ, เป็นพยางค์หน้าของคำ ซึ่งออกเสียง กระ ตระ ประ พระ ไม่ว่าภาษาใดๆ เช่น กระติก กระถาง ตระกูล ตระกอง ประสาท ประณีต พระหาวัน พระหาสุข ฯลฯ

- **ลดรูป** คือไม่ต้องประวิสรรชนีย์ (ะ) ในพยางค์ที่ออกเสียง อะ แต่ต้องออกเสียงเป็น อะ เหมือนมีวิสรรชนีย์กำกับอยู่ ในกรณีต่อไปนี้ คือ เป็นพยางค์ที่เป็นอักษรนำ เช่น ขนม ขยัน ถนอม ผนวช ฯลฯ, เป็นคำยกเว้น ซึ่งใช้เขียนไม่มีประวิสรรชนีย์ เช่น ณ ที่แปลว่า ในที่, เป็นตัวสะกดในคำไทยบางคำซึ่งนิยมออกเสียงตัวสะกดเป็นพิเศษ เช่น สกปรก จักจั่น สัพยอก, เป็นพยางค์เดิมที่แผลงมาจากคำเดิมที่ไม่มีวิสรรชนีย์หรือไม่ประวิสรรชนีย์, เป็นพยางค์ที่ออกเสียงเบา เช่น ขโมย ชนวน สไบ, เป็นพยางค์ที่มิใช่อยู่สุดท้ายของคำบาลีสันสกฤตและคำในภาษาตระกูลอินเดีย-ยุโรป เช่น คติ อมรินทร์ อคติ
- **แปลงรูป** คือแปลงวิสรรชนีย์เป็น ไม้หันอากาศ ในเมื่อมีตัวสะกดเช่น กัน = กะ + น, อักษรหัน คือเปลี่ยนวิสรรชนีย์ให้เป็นพยัญชนะตัวเดียวกับตัวสะกด ปัจจุบันใช้อยู่ตัวเดียวคือ ร หัน
- **ตัดรูปและตัดเสียง** เช่น อนุช เป็น นุช , อดิเรก เป็น ดิเรก

5.3.2 สระอึ มีวิธีใช้ดังนี้

- **ต้องมี อ กำกับซึ่งเดิมเข้ามา** เมื่อใช้ในพยางค์ที่ไม่มีตัวสะกด
- **ไม่ต้องมี อ กำกับ** เมื่อใช้ในพยางค์ที่มีตัวสะกด

5.3.3 สระเอะ แอะ มีวิธีใช้ดังนี้

- **คงรูป** เช่น เละ เฟะ แพะ และ
- **แปลงรูป** คือแปลงวิสรรชนีย์เป็นไม้ไต่คู้เช่น เก็ง เพ็ง แข็ง แม็ก

5.3.4 สระโอะ มีวิธีใช้ดังนี้

- **คงรูป** ใช้ทั่วไป เมื่อไม่มีตัวสะกด เช่น โกะ โต๊ะ โผละ
- **ลดรูป** คือตัดรูปสระโอะออกทั้งหมด คงไว้แต่พยัญชนะที่ถูกสะกด เช่น กง คน ขน

5.3.5 สระเอาะ มีวิธีใช้ดังนี้

- **คงรูป** เช่น เกาะ เบาะ เพาะ

- **ลดรูป** คือตัดรูปเดิมออกทั้งหมดแล้วใช้ตัว อ กับไม้ไต่คู้แทน เช่น น็อต ล็อก

5.3.6 สระออ มีวิธีใช้ดังนี้

- **คงรูป** คือต้องมีตัว อ กำกับอยู่ด้วยเสมอ เช่น ขอ พอ พ่อ หลอ
- **ลดรูป** คือตัดตัว อ ออกเสีย แต่คงอ่านเหมือนมีตัว อ กำกับอยู่ ได้แก่ คำไทยบางคำ คือ บ และ บ่ ที่แปลว่าไม่ แต่ถ้า บอ ที่แปลว่าเกือบบ้า หรือ บ่อ ที่แปลว่าหลุม ต้องมีตัว อ อยู่ด้วย, คำบาลีสันสกฤตและคำในภาษาอื่นบางคำที่ใช้ ร เป็นตัวสะกด เช่น กร พร จร สมร ขจร, คำบาลีสันสกฤตที่พยางค์หน้าเป็น ป แต่แผลงเป็น บ เช่น บดี บพิตร บรม บรมัตถ์ บวร เป็นต้น

5.3.7 สระเออ มีวิธีการใช้ดังนี้

- **คงรูป** วิธีนี้ใช้ไรพยางค์ทั่วไปที่ไม่มีตัวสะกด เช่น เธอ เกลอ เผอเรอ
- **ลดรูป** คือลดตัว อ ออกเสีย ในเมื่อมีตัวสะกดในแม่เกย (ตัว ย สะกด) เช่น เชย เลย
- **แปลงรูป** คือแปลงตัว อ เป็น อิ ในเมื่อมีตัวสะกด เช่น เดิน เพิก เพลิง

5.3.8 สระเอีย มีวิธีใช้ดังนี้

- **คงรูป** คือไม่เปลี่ยนรูป จะมีตัวสะกดหรือไม่ก็ตาม เช่น เสีย เลีย เขียน เรียน ฯลฯ
- **ลดรูป** คือตัดไม้หน้ากับพินทุ อี ออก แต่ปัจจุบันเลิกใช้แล้ว

5.3.9 สระอัวะ มีวิธีใช้ดังนี้

- **คงรูป** เช่น ผัวะ ยัวะ ฯลฯ
- **ลดและแปลงรูป** คือลดหันอากาศแล้วแปลงวิสรรชนีย์เป็นไม้ไต่คู้ ในเมื่อมีตัวสะกด เช่น ก็วง ข็วก

5.3.10 สระอัว มีวิธีใช้ดังนี้

- **คงรูป** เช่น มัว กลัว ผัว ฯลฯ ถ้าเขียนคำในภาษาบาลี หรือสันสกฤต ให้ออกเสียงเป็น “เอา”
- **ลดรูป** คือลดหันอากาศ ในเมื่อมีตัวสะกด เช่น กวน ควง รวบ สวย
- **แปลงรูป** คือแปลงหันอากาศเป็นตัว ว ทำนองเดียวกับอักษรหัน แต่แปลกกันที่ไม่มีตัวสะกด เช่น หวว = หัว แต่ไม่เห็นใช้ในปัจจุบัน

5.3.11 สระฤ มีวิธีใช้ 3 อย่างคือ

- **ใช้โดดๆ** แปลว่า หรือ ไม่ มักจะใช้ในคำประพันธ์
- **ใช้เป็นพยางค์หน้าหรือพยางค์หลังของคำที่มาจากสันสกฤต** เช่น ฤดู หฤทัย นฤบดี

- **ใช้ประสมกับพยัญชนะ** คล้ายกับตัว ร ควบ เช่น ทฤษฎี คฤหาสน์ พฤศจิกายน

5.3.12 สระฤา มีวิธีใช้ดังนี้

- **ใช้โดดๆ** ไม่ประสมกับพยัญชนะ แปลว่า หรือ อะไร ไม่ ไม่ใช่ ปัจจุบันใช้แต่ในคำประพันธ์เท่านั้น เช่น ฤาเบา = ไม่เบา ไม่ใช่น้อย
- **ใช้เป็นพยางค์หน้าของคำ** เช่น ฤาดี = ยินดี , ฤาสาย = เชื้อสายที่เลื่องลือ

5.3.13 สระใอ ไม้ม้วน การใช้สระใอ ไม้ม้วน และสระไอไม้มลาย ในสมัยก่อนไม่สู้จะพิถีพิถันนัก ท่านเพ่งเล็งถึงเสียงมากกว่ารูป เพราะฉะนั้นจึงใช้สลับสับสนกัน ต่อมาท่านได้วางหลักเกณฑ์ให้ใช้ไม้ม้วนได้เพียง 20 คำเท่านั้น ได้แก่ ใกล้ ใคร ใคร่ ใจ ใช่ ใช้ ใด ใต้ ใน ใบ ใบ้ ใฝ่ ใย สะใภ้ ใส ใส่ ให้ ใหญ่ ใหม่ ไหล

5.3.14 สระไอ ไม้หลาย นอกจากคำ 20 คำที่บังคับให้ใช้ไม้ม้วนแล้ว บรรดาคำไทยที่ออกเสียง “ไอ” ให้ใช้สระไอไม้มลาย ทั้งสิ้น ส่วนคำที่มาจากบาลีสันสกฤต พยางค์ที่ออกเสียง “ไอ” มีวิธีเขียนถึง 4 วิธีคือ ไอ ไอย อัย อัยย

5.3.15 วิธีใช้สระโดยสรุป

- **คงรูป** คือต้องเขียนรูปให้ปรากฏชัดเช่น :- กะบะ กะปิ ดีนี่ ไปไหน ทำไม นานโข
- **ลดรูป** คือไม่ต้องเขียนรูปสระให้ปรากฏหรือปรากฏเพียงบางส่วน แต่ต้องออกเสียงให้ตรงกับรูปสระที่ลดนั้น การลดรูปมี 2 อย่างคือ :-
  - ลดรูปทั้งหมด ได้แก่ พยัญชนะ + สระโอ + ตัวสะกด(ยกเว้นตัว ร) เช่น :-

    น + โอะ + ก = นก ม + โอะ + ด = มด

    ก + ออ + ร-สะกด = กร จ + ออ + ร-สะกด = จร
  - ลดรูปบางส่วน ได้แก่สระที่ลดรูปไม่หมด เหลือไว้เพียงบางส่วนของรูปเต็มพอเป็นเครื่องสังเกตให้รู้ว่าไม่ซ้ำกับรูปอื่นเช่น :-

    ก + เออ + ย-สะกด = เกย (ลดรูปตัว อ เหลือแต่ไม้หน้า)

    ก + อัว + น-กด = กวน (ลดหันอากาศ เหลือแต่ตัว ว)
- **แปลงรูป** คือแปลงสระรูปเดิมให้เปลี่ยนเป็นอีกรูปหนึ่งเช่น :-

  ก + อะ + น-สะกด = กัน (แปลงวิสรรชนีย์เป็นหันอากาศ)

  ก + เอะ + ง-สะกด = เก็ง (แปลงวิสรรชนีย์เป็นไม้ไต่คู้)

  ข + แอะ + ง-สะกด = แข็ง (แปลงวิสรรชนีย์เป็นไม้ไต่คู้)
- **ตัดรูป** คือตัดสระ อะ ที่เป็นสระหน้าของคำที่มาจากบาลีและสันสกฤต และไม่ต้องออกเสียงสระที่ตัดนั้น (คือตัดทั้งรูปทั้งเสียง) เช่น :- อนุช-นุช, อดิเรก-ดิเรก, อภิปราย-ภิปราย, อภิรมย์-ภิรมย์

- เติมรูป คือเพิ่มรูปเข้ามานอกเหนือจากที่มีอยู่แล้ว ได้แก่สระ อี ที่ใช้ในมาตรา ก กา เช่น ม + อี = มี เติม อ เป็น มือ , ค + อี = คี เติม อ เป็น คือ

  เหตุที่เติมรูปเพราะในการเขียนรูปสระ อี และ อื ในสมัยโบราณ ไม่ตัวพิมพ์ใช้ ต้องใช้เขียน ถ้าเขียนตัวบรรจงก็สังเกตง่าย ไม่มีปัญหา, แต่ถ้าเขียนหวัดก็ทำให้ฉงนเช่น มี กับ มื มีความหมายด้วยกันทั้งสอง ยากแก่การวินิจฉัยว่าควรจะเป็น มี หรือ มือ แต่ถ้าเติมรูป อ ลงไปที่สระอี ก็เข้าใจได้ว่าเป็นสระออ , ส่วนสระ อี คงรูปไว้อย่างเดิมก็จะทำให้การอ่านง่ายขึ้น ถึงแม้จะเขียนสระ อื ตกรูปฝนทองไปหนึ่งอันเป็น มือ ก็รู้ได้ว่าเป็นสระ อื เพราะมีตัว อ กำกับอยู่จะออกเสียงเสียงเป็นสระ อี ไม่ได้ การเติมรูปสระจึงมีประโยชน์ดังนี้

หมายเหตุ :-

1. สระบางตัวอาจใช้ทั้งวิธีลดรูปและแปลงรูปก็ได้เช่น :- ก + เอาะ = เกาะ แล้วลดรูป ไม้หน้ากับ ลากข้าง และแปลงรูปวิสรรชนีย์เป็นไม้ไต่คู้ ก็จะกลายรูปเป็น ก็
2. สระที่ประสมกับวิสรรชนีย์บางตัว เมื่อแปลงวิสรรชนีย์เป็นไม้ไต่คู้แล้ว จะกลับเป็นสระตัวเดิมที่ยังมิได้ประสมกับวิสรรชนีย์เช่น เอาะ จะกลับเป็น ออ ในเมื่อแปลงวิสรรนีย์เป็นไม้ไต่คู้ และมีตัวสะกด ตัวอย่าง :- ก + เอาะ = เกาะ + ก-สะกด จะเป็น ก็อก เมื่อแปลงวิสรรชนีย์เป็นไม้ไต่คู้ แล้วสระ เอาะ ก็จะกลับเป็นสระ ออ ตัวเดิมที่ยังมิได้ประสมกับวิสรรชนีย์จึงจะมีรูปเป็น ก็อก

5.4 พยัญชนะ

พยัญชนะ แปลว่า กระทำเสียงให้ปรากฏชัดเจน , ในภาษาพูดหมายถึง สำเนียงชัดเจนที่เกิดจากพยัญชนะประสมกับสระ แต่อาจจะมีความหมายหรือไม่ก็ได้ , ในภาษาหนังสือ หมายถึง เครื่องหมายหรือตัวอักษรที่สำหรับใช้ประสมกับสระ เพื่อใช้แทนภาษาพูดหรือจารึกคำพูดไว้มิให้สูญ ตามหลักภาษาพยัญชนะจะออกเสียงตามลำพังไม่ได้ จะต้องมีรูปสระประสมอยู่ด้วยจึงจะออกเสียงได้ และสระที่ประสมอยู่นั้น จะเป็นสระปรากฏรูปหรือลดก็ได้ ในภาษาบาลีมีชื่อสำหรับเรียกพยัญชนะอีกอย่างหนึ่งว่า “นิสิต” หมายความว่า ต้องอาศัยสระจึงออกเสียงได้ เพราะฉะนั้น สระกับพยัญชนะ ตามปรกติจะต้องใช้คู่กันเสมอ โดยเฉพาะในภาษาไทย จะใช้เขียนสระตามลำพังโดยไม่มีพยัญชนะกำกับไม่ได้ เว้นไว้แต่รูปสระบางตัวที่เราถ่ายแบบมาจากสันสกฤต คือ ฤ ฤา ฦ ฦา

พยัญชนะในภาษาไทยก็แบ่งออกเป็นวรรค ๆ ตามฐานที่เกิดอักษร เหมือนในภาษาบาลีและสันสกฤต คือ จัดพยัญชนะที่เกิดจากฐานต่างๆ ดังนี้

เกิดจากฐานคอ (กัณฐชะ)	: ก ข ฃ ค ฅ ฆ ง	เรียกว่าวรรค ก
เกิดจากฐานเพดาน (ตาลุชะ)	: จ ฉ ช ฌ ฌ ญ	เรียกว่าวรรค จ
เกิดจากฐานปุ่มเหงือก (มุทธชะ)	: ฏ ฏ ฐ ฑ ฒ ณ	เรียกว่าวรรค ฏ
เกิดจากฐานฟัน (ทันตชะ)	: ด ต ถ ท ธ น	เรียกว่าวรรค ด
เกิดจากฐานริมฝีปาก (โอฐชะ)	: บ ป ผ ฝ พ ฟ ภ ม	เรียกว่าวรรค บ
เกิดจากฐานต่าง ๆ กัน	: ย ร ล ว ศ ษ ส ห ฬ อ ฮ	เรียกว่าเศษวรรค

5.5 การใช้พยัญชนะ

พยัญชนะในภาษาไทย จำแนกวิธีใช้ออกเป็น ชนิด มาตรา คำเป็นคำตาย ไตรยางค์ อักษรคู่ อักษรเดี่ยว อักษรประสม ตัวสะกด ตัวหัน และการันต์

5.5.1 ชนิดของพยัญชนะ

พยัญชนะ 44 ตัวจำแนกออเป็น 3 ชนิดคือ :-

1. พยัญชนะที่ใช้เขียนได้ทั่วไปทั้งคำไทย คำบาลี คำสันสกฤต และคำในภาษาอื่นที่เรานำมาใช้ในภาษาไทย เช่น เขมร มอญ ชวา อังกฤษ เป็นต้น มีทั้งหมด 21 ตัว คือ ก ข ค ง จ ฉ ช ต ถ ท น ป ผ พ ม ย ร ล ว ส ห รวมเรียกว่า พยัญชนะกลาง
2. พยัญชนะที่ใช้เขียนในกรอบจำกัด เฉพาะคำที่มาจากบาลีสันสกฤต และภาษายุโรปบางคำ มี 13 ตัว คือ ฆ ฌ ญ ฏ ฐ ฑ ฒ ณ ธ ภ ศ ษ ฬ รวมเรียกว่า พยัญชนะเดิม หรือ พยัญชนะเฉพาะ

พยัญชนะเหล่านี้ ไทยเราตั้งขึ้นสำหรับใช้เขียนแทนตัวอักษรในภาษาเดิม ซึ่งมีสำเนียงเป็น อุรชะ (เกิดจากฐานอก ที่เรียกว่า “ลงทรวง”) ได้แก่ตัว ฆ ฌ ฒ ธ ภ, มีสำเนียงเป็น นาสิกย์ (เกิดจากฐานจมูก) ได้แก่ตัว ญ ณ, มีสำเนียงเป็น มุทธชะ (เกิดจากฐานปุ่มเหงือก) ได้แก่ตัว ฏ ฐ ฑ ณ ฬ (ตัว ณ เป็นทั้ง มุทธชะ และ นาสิกย์), มีสำเนียงเป็น อุสุม (มีเสียงเสียดแทรกออกทางฟัน) ได้แก่ตัว ศ ษ

เสียงของพยัญชนะ 13 ตัวนี้ ไม่มีในภาษาไทย เราจึงไม่นิยมใช้เขียนคำไทยทั่วไป เพราะเราตั้งขึ้นเพียงเพื่อให้ครบตามจำนวนตัวอักษรในภาษาเดิมเท่านั้น แต่ถึงกระนั้นก็ดี ยังมีบางตัวที่เรานำมาใช้เขียนคำไทยบ้างบางคำ ทั้งนี้เนื่องจากเขียนติดมาแต่โบราณ จนกลายเป็นความนิยม และเป็นที่รับรองว่าถูกต้องตามหลักภาษาซึ่งต้องนับว่าอยู่ในข้อยกเว้นพิเศษ

3. พยัญชนะที่ไทยบัญญัติเพิ่มขึ้นเพื่อให้พอแก่การเขียนสำเนียงในภาษาไทย เพราะพยัญชนะเท่าที่อยู่ในบาลี และ สันสกฤต ไม่สามารถจะเขียนสำเนียงในภาษาไทยได้ครบทุกเสียง จึงจำเป็นต้องบัญญัติเพิ่มเติมขึ้นอีก 10 ตัว คือ ฃ ฅ ฌ ฎ ด บ ผ ฟ อ ฮ รวมเรียกว่า พยัญชนะเติม

5.5.2 มาตรา

มาตรา คือแม่บทแลกลูกอักษร ตามหมวดคำที่มีตัวกะกด หรือออกเสียงอย่างเดียวกัน แบ่งเป็น 8 มาตรา หรือ 8 แม่ คือ :-

- มาตรา ก กา หรือ แม่ ก กา คือพยางค์ที่ไม่มีตัวสะกด
- มาตรา กน หรือ แม่ กน คือพยางค์ที่มีตัว น สะกดหรือตัวอื่นซึ่งทำหน้าที่เหมือนตัว น สะกด ได้แก่ตัว น ญ ณ ร ล ฬ
- มาตรา กง หรือ แม่กง คือพยางค์ที่มีตัว ง สะกดหรือนิคหิตซึ่งทำหน้าที่เหมือนตัว ง สะกด
- มาตรา กก หรือ แม่ กก คือพยางค์ที่มีตัว ก สะกดหรือตัวอื่นซึ่งทำหน้าที่เหมือนตัว ก สะกด ได้แก่ตัว ก ข ค ฆ
- มาตรา กด หรือ แม่ กด คือพยางค์ที่มีตัว ด สะกดหรือตัวอื่นซึ่งทำหน้าที่เหมือนตัว ด สะกด ได้แก่ตัว ด จ ฉ ช ซ ฌ ฎ ฏ ฐ ฑ ฒ ต ถ ท ธ ศ ษ ส
- มาตรา กบ หรือ แม่ กบ คือพยางค์ที่มีตัว บ สะกดหรือตัวอื่นซึ่งทำหน้าที่เหมือนตัว บ สะกด ได้แก่ตัว บ ป พ ฟ ภ
- มาตรา กม หรือ แม่ กม คือพยางค์ที่มีตัว ม สะกดหรือนิคหิตซึ่งทำหน้าที่เหมือนตัว ม สะกด
- มาตรา เกย หรือ แม่ เกย คือพยางค์ที่มีตัว ย และตัว ว สะกด

5.5.3 คำเป็นคำตาย

พยางค์ที่ปรากฏในแม่ต่างๆ นั้นมีทั้งเสียงหนัก เบา ยาว สั้น บางพยางค์ก็เป็นคำเป็น บางพยางค์ก็เป็นคำตาย มีหลักสังเกตให้รู้ว่าพยางค์ไหนเป็นคำเป็นหรือคำตายดังนี้ :-

คำเป็น คือ พยางค์ที่ประสมด้วยสระเสียงยาวในแม่ ก กา และพยางค์ที่มีตัวสะกดในแม่ กน กง กม เกย พยางค์ที่ประสมด้วยสระ อำ ใอ ไอ เอา ซึ่งเป็นสระเสียงสั้นก็นับเป็นคำเป็นด้วย

คำตาย คือ พยางค์ที่ประสมด้วยสระเสียงสั้นในแม่ ก กา (ยกเว้นเสียงสั้น 4 ตัว คือ อำ ใอ ไอ เอา) และพยางค์ที่มีตัวสะกดในแม่ กก กด กบ

5.5.4 ไตรยางค์

ไตรยางค์ คืออักษร 3 หมู่ซึ่งจัดแยกออกมาเป็นพวกๆ จากพยัญชนะ 44 ตัว ได้แก่ อักษรสูง อักษรกลาง อักษรต่ำ

อักษรสูงมี 11 ตัวคือ ข ฃ ฉ ฐ ถ ผ ฝ ศ ษ ส ห

อักษรกลางมี 9 ตัวคือ ก จ ฎ ฏ ด ต บ ป อ

อักษรต่ำมี 24 ตัวคือ ค ฅ ฆ ง ช ซ ฌ ญ ฑ ฒ ณ ท ธ น พ ฟ ภ ม ย ร ล ว ฬ ฮ

การที่จัดแยกพยัญชนะออกเป็นอักษร 3 หมู่ (ไตรยางค์) นั้น ก็โดยถือเอาเสียงเป็นสำคัญ คือ พยัญชนะตัวใดพื้นเสียงที่ยังมิได้ผันด้วยรูปวรรณยุกต์ มีสำเนียงอยู่ในระดับสูง ก็จัดเป็นพวกอักษรเสียงสูง , พยัญชนะตัวใดพื้นเสียงที่ยังมิได้ผันด้วยรูปวรรณยุกต์ มีสำเนียงอยู่ในระดับกลางก็จัดเป็นพวกอักษรเสียงกลาง, พยัญชนะตัวใด พื้นเสียงที่ยังมิได้ผันด้วยรูปวรรณยุกต์ มีสำเนียงอยู่ในระดับต่ำ ก็จัดเป็นพวกอักษรเสียงต่ำ ที่เรียกว่าอักษรต่ำน่าจะหมายถึงเสียงต่ำกว่าอักษร 2 พวกข้างต้น

การที่จำแนกพยัญชนะออกเป็นอักษรสูง อักษรกลาง อักษรต่ำนั้นนับว่าเป็นประโยชน์มากคือ :-

1. สามารถผันคำให้มีเสียงและรูปต่างๆ เมื่อเสียงและรูปต่างกันความหมายก็ต่างกันกันด้วย เช่น ไข ไข่ ไข้ ย่อมแสดงความหมายคลี่คลายไปจากคำเดิมเช่นเดียวกับ เสือ เสื่อ เสื้อ ทั้งนี้นับว่าเป็นแนวทางสำคัญที่ทำให้ภาษาขยายตัว และแสดงความประณีตในการเขียนอักษรไทย ซึ่งสามารถจะใช้บันทึกเสียงได้ใกล้ชิดเป็นอย่างดี
2. สามารถนำคำบาลีและสันสกฤตมาเข้าแนวสำเนียงของคนไทยได้สนิทสนม เช่น เล่ห์ สมุทร สนเท่ห์ พุทโธ่ สมัคร ฯลฯ
3. ไม่ต้องเขียนเครื่องหมายวรรณยุกต์กำกับลงไปทุกคำ เช่น “ขา” ไม่ต้องเขียนเป็น “ค่า” ทั้งนี้นับว่าช่วยให้การเขียนหนังสือสะดวกและรวดเร็วยิ่งขึ้น

5.5.5 วรรณยุกต์

อักษร 3 หมู่นั้นใช้ผันด้วยวรรณยุกต์ วรรณยุกต์มี 2 ชนิดคือ :-

1. วรรณยุกต์มีรูป หมายถึงวรรณยุกต์ที่มีเครื่องบอกระดับของเสียงให้เห็นชัดเจนอยู่บนอักษร มีอยู่ 4 รูปคือ ่ ้ ๊ ๋ เรียกชื่อว่าวรรณยุกต์ เอก โท ตรี จัตวา โดยลำดับ และให้เขียนไว้เบื้องบนอักษรตอนสุดท้าย เช่น :- ก่ ก้ ก๊ ก๋ ปั่น ลื่น เลี่ยน เป็นต้น ถ้าเป็นอักษรควบหรืออักษรนำให้เขียนไว้เบื้องบนอักษรตัวที่ 2 เช่น :- ครุ่น คลื่น เกลื่อน เกล้า ฯลฯ
2. วรรณยุกต์ไม่มีรูป ได้แก่เสียงที่มีทำนองสูงต่ำตามหมู่ของอักษร โดยไม่ต้องมีรูปวรรณยุกต์กำกับก็อ่านออกเสียงได้เหมือนมีรูปวรรณยุกต์กำกับอยู่ด้วย

วรรณยุกต์ไม่มีรูปต่างกับวรรณยุกต์มีรูปคือ วรรณยุกต์มีรูปจะต้องมีเครื่องหมายบอกเสียงกำกับไว้ข้างบนอักษร และมีอยู่เพียง 4 เสียงเท่านั้นคือ เสียงเอก เสียงโท เสียงตรี เสียงจัตวา ตามรูปวรรณยุกต์แต่ไม่

มีเสียงสามัญ ส่วนวรรณยุกต์ไม่มีรูปมีครบ 5 เสียงเต็มตามจำนวนเสียงที่กำหนดใช้อยู่ในภาษาไทย โดยไม่มีเครื่องหมายบอกเสียงกำกับ แต่ออกเสียงสูงต่ำตามหมู่ของอักษรทั้ง 3

5.5.6 วิธีผันอักษร 3 หมู่

อักษร 3 หมู่ที่เรียกว่า ไตรยางค์ นั้นใช้ผันด้วยรูปวรรณยุกต์ต่างๆ กันดังนี้ :-

อักษรสูง ผันด้วยวรรณยุกต์ ' ้ คำเป็นผันได้ 3 คำ คำตายผันได้ 2 คำ ดังนี้ :-

คำเป็น พื้นเสียงเป็นจัตวา	เช่น :-	ขา ขง ขน ขม เขย ขาว
ผันด้วย ' เป็นเสียงเอก	”	ข่า ข่ง ข่น ข่ม เข่ย ข่าว
ผันด้วย ้ เป็นเสียงโท	”	ข้า ข้ง ข้น ข้ม เข้ย ข้าว
คำตาย พื้นสียงเป็นเอก	เช่น :-	ขะ ขก ขด ขบ ขาก ขาด ขาบ
ผันด้วย ้ เป็นเสียงโท	”	ข้ะ ข้ก ข้ด ข้บ ข้าก ข้าด ข้าบ

อักษรกลาง ผันด้วยวรรณยุกต์ ' ้ ๊ ๋ คำเป็นผันได้ 5 คำ คำตายผันได้ 4 คำ ดังนี้ :-

คำเป็น พื้นเสียงเป็นสามัญ	เช่น :-	กา กง กน กม เกย กาว
ผันด้วย ' เป็นเสียงเอก	”	ก่า ก่ง ก่น ก่ม เก่ย ก่าว
ผันด้วย ้ เป็นเสียงโท	”	ก้า ก้ง ก้น ก้ม เก้ย ก้าว
ผันด้วย ๊ เป็นเสียงตรี	”	ก๊า ก๊ง ก๊น ก๊ม เก๊ย ก๊าว
ผันด้วย ๋ เป็นเสียงจัตวา	”	ก๋า ก๋ง ก๋น ก๋ม เก๋ย ก๋าว
คำตาย พื้นเสียงเป็นเอก	เช่น :-	กะ กก กด กบ กาก กาด กาบ
ผันด้วย ้ เป็นเสียงโทร	”	ก้ะ ก้ก ก้ด ก้บ ก้าก ก้าด ก้าบ
ผันด้วย ๊ เป็นเสียงตรี	”	ก๊ะ ก๊ก ก๊ด ก๊บ ก๊าก ก๊าด ก๊าบ
ผันด้วย ๋ เป็นเสียงจัตวา	”	ก๋ะ ก๋ก ก๋ด ก๋บ ก๋าก ก๋าด ก๋าบ
คำตายสระยาวผันเช่นเดียวกับสระสั้น		

อักษรต่ำ ผันด้วยวรรณยุกต์ ' ้ ๋ คำเป็นผันได้ 3 คำ ใช้วรรณยุกต์ ' ้ ดังนี้ :-

คำเป็น พื้นเสียงเป็นสามัญ	เช่น :-	คา คง คน คม เคย คาว
ผันด้วย ' เป็นเสียงโท	”	ค่า ค่ง ค่น ค่ม เค่ย ค่าว
ผันด้วย ้ เป็นเสียงตรี	”	ค้า ค้ง ค้น ค้ม เค้ย ค้าว

คำตาย ผันได้ 3 คำใช้วรรณยุกต์ ' ้ ๋ แบ่งออกเป็น 2 ชนิดดังนี้ :-

คำตายสระสั้น พื้นเสียงเป็นตรี	เช่น :-	คะ คก คด คบ
ผันด้วย ' เป็นเสียงโท	”	ค่ะ ค่ก ค่ด ค่บ
ผันด้วย ๋ เป็นเสียงจัตวา	”	ค๋ะ ค๋ก ค๋ด ค๋บ

คำตายสระยาว พื้นเสียงเป็นโท	เช่น :-	คาก คาด คาบ
ผันด้วย ้ เป็นเสียงตรี	”	ค้าก ค้าด ค้าบ
ผันด้วย ๊ เป็นเสียงจัตวา	”	ค๊าก ค๊าด ค๊าบ

จะเห็นได้ว่า อักษรสูงกับอักษรกลางมีเสียงตรงกับรูปวรรณยุกต์เสมอ แต่อักษรต่ำจะมีเสียงสูงกว่ารูปวรรณยุกต์หนึ่งขั้น เว้นไว้แต่วรรณยุกต์จัตวา ซึ่งเสียงคงเป็นจัตวาตามรูปวรรณยุกต์เพราะไม่มีเสียงใดที่สูงไปกว่านั้นอีก

5.5.7 อักษรคู่-อักษรเดี่ยว

อักษรต่ำ 24 ตัว แบ่งออกเป็น 2 พวก คือ :-

1. พวกที่มีเสียงคู่กับอักษรสูง สามารถร่วมเสียงกับอักษรสูง แล้วผันให้ครบเสียงวรรณยุกต์ทั้ง 5 ได้เหมือนอักษรกลาง อักษรพวกนี้มีชื่อเรียกว่า อักษรคู่ มี 14 ตัวคือ ค ฅ ฆ ช ซ ฌ ฑ ฒ ท ธ พ ฟ ภ ฮ

อักษรคู่ 14 ตัวนี้ จัดเป็นคู่อักอักษรสูงตามสำเนียงที่ใกล้ชิดกัน และเกิดจากฐานเดียวกันดังนี้ :-

ค ฅ ฆ	คู่กับ	ข ฃ
ช ฌ	”	ฉ
ซ	”	ศ ษ ส
ฑ ฒ ท ธ	”	ฐ ถ
พ ภ	”	ผ
ฟ	”	ฝ
ฮ	”	ห

เมื่อนำแต่ละคู่ไปผันรวมกันและเรียงเสียงให้เข้าระดับกันแล้ว จะได้เสียงวรรณยุกต์ครบทั้ง 5 ตัวอย่างเช่น

	สามัญ	เอก	โท	ตรี	จัตวา
คู่ที่ 1	ค	ข่	ค่-ข้	ค้	ข
คู่ที่ 2	ช	ฉ่	ช่-ฉ้	ช้	ฉ
คู่ที่ 3	ซ	ส่	ซ่-ส้	ซ้	ส
คู่ที่ 4	ท	ถ่	ท่-ถ้	ท้	ถ
คู่ที่ 5	พ	ผ่	พ่-ผ้	พ้	ผ
คู่ที่ 6	ฟ	ฝ่	ฟ่-ฝ้	ฟ้	ฝ
คู่ที่ 7	ฮ	ห่	ฮ่-ห้	ฮ้	ห

2. พวกที่มีเสียงเดี่ยว คือ จะใช้เสียงคู่กับอักษรสูงไม่ได้ เนื่องจากมีแนวเสียงต่างกัน เพราะฉะนั้น จึงได้ชื่อว่า อักษรเดี่ยว มี อยู่ 10 ตัว คือ :- ง ญ ณ น ม ย ร ล ว ฬ

เพราะเหตุที่อักษรเดี่ยว 10 ตัวเป็นอักษรที่มีเสียงเบา และบางตัวก็ทำหน้าที่เป็นกึ่งสระ จึงอาจออกเสียงประสมกับพยัญชนะที่มานำข้างหน้าได้ แต่การออกเสียงนั้น บางทีก็ประสมกันสนิท เช่น :- ครู ปลา กวาง ฯลฯ บางทีก็จะประสมกันไม่สนิท เช่น :- สง่า ขนม ฉลอง ฯลฯ อักษรที่จะใช้นำนั้นอาจเป็นอักษรสูง กลาง หรือต่ำ ก็ได้ แต่มีหลักอยู่ว่าจะต้องผันตามหลักการผันของตัวนำเสมอ คือ ถ้าอักษรสูงนำก็ต้องผันตามอักษรสูง ถ้าอักษรกลางนำก็ต้องผันตามอักษรกลาง อักษรต่ำก็ผันเสียงตามอักษรต่ำ และเป็นที่สังเกตว่าแม้อักษรเดี่ยวจะไม่มีเสียงอักษรสูงมาเป็นเสียงคู่ไม่ได้ แต่ก็อาจผันให้ครบเสียงวรรณยุกต์ทั้ง 5 ได้ โดยใช้อักษรกลาง หรืออักษรสูงเป็นตัวนำ

5.5.8 อักษรประสม

อักษรประสม คือ พยัญชนะ 2 ตัวควบกันและร่วมสระเดียวกัน บางคำก็ออกเสียงพร้อมกันเป็นอักษรกล้ำ บางคำก็ออกเสียงคล้ายกับเป็น 2 พยางค์ บางคำก็ออกเสียงแปรไปเป็นพยัญชนะตัวอื่น ไม่ตรงกับเสียงพยัญชนะตัวจริงที่ปรากฏรูป

อักษรประสม แบ่งออกเป็น 2 ชนิดคือ อักษรควบ และอักษรนำ

5.5.8.1 อักษรควบ

อักษรควบ คือ พยัญชนะ2 ตัวควบหรือกล้ำอยู่ในสระเดียวกัน โดยลักษณะของอักษรควบมีดังนี้

1. มีตัว ร ล ว ห เป็นหลัก
2. ร จะเรียงไว้ข้างหน้า หรือ ข้างหลังตัวอื่นที่มาควบก็ได้ เช่น : ครู ใคร ตรา (เรียงไว้ข้างหลัง) , มารค บรรพ มรรค สามารถ (เรียงไว้ข้างหน้า) แต่ตัว ล ว จะเรียงไว้ข้างหลังตัวอื่นที่มาควบเสมอ เช่น : คลอง พลาด กวาด ไขว่ เป็นต้น ส่วนตัว ห จะอยู่หน้า ม เสมอ
3. ตัว ร เมื่อควบกับตัวอื่น บางทีก็ออกเสียง เช่น : ครั่นคร้าม ครึกโครม ฯลฯ แต่บางทีก็มิออกเสียง เช่น : จริง เศร้า สรวม ไซร้ ฯลฯ แต่ตัว ล ว ออกเสียงเสมอ

อักษรควบแบ่งออกเป็น 2 ชนิด คือ :-

ก. อักษรควบแท้ คือ อักษรควบที่เกิดจากพยัญชนะ 2 ตัวควบ หรือ กล้ำอยู่ในสระเดียวกัน ได้แก่พยัญชนะที่ควบหรือกล้ำกับตัว ร ล ว เมื่อควบหรือกล้ำแล้วก็ต้องออกเสียงพร้อมกัน เป็นตัวสะกด หรือ การันต์ต้องเป็นด้วยกัน

1. การออกเสียงพร้อมกัน เช่น :- ไกร ครู คลา พลอด ไกว ขวา ควาน ฯลฯ

2. เป็นตัวสะกดด้วยกัน เช่น :- บุตร เนตร มิตร จิตร กอปร ฯลฯ ตัวสะกดเหล่านี้ ถ้ามีรูปสระกำกับ ก็ทำหน้าที่เป็นทั้งตัวสะกด และออกเสียงด้วยในคราวเดียวกัน เช่น :- บุตรี อ่านว่า บุด-ตรี, จิตรา อ่านว่า จิด-ตรา, จักรี อ่านว่า จัก-กรี ฯลฯ
3. เป็นตัวการันต์ด้วยกัน เช่น :- พักตร์ วัสตร์ ศาสตร์ อินทร์ ฯลฯ

ข. อักษรควบไม่แท้ คือ อักษรควบที่เกิดจากพยัญชนะ 2 ตัวควบหรือกล้ำอยู่ในสระตัวเดียวกัน ได้แก่พยัญชนะที่ควบกับตัว ร และตัว ร นั้นจะอยู่หน้าหรือกลังก็ได้ แต่ออกเสียงพยัญชนะตัวหน้าหรือตัวหลังเพียงตัวเดียว หรือบางตัวก็ออกเสียงแปรเปลี่ยนไปเป็นเสียงพยัญชนะตัวอื่น และจะเป็นตัวสะกดด้วยกันก็ได้ หรือจะแยกให้ตัวหน้าเป็นตัวสะกดแต่ตัวเดียว และให้ตัวหลังเป็นการันต์ก็ได้ ตัวอย่างเช่น

1. ออกเสียงเฉพาะตัวหน้า :- จริง ไซร้ ศรัทธา ปราศรัย ศราทธ์ ศรี เศร้า สรง สรวง สรวม สรวล สร้อย สระ สร่าง สร้าง สราท สร้าว เสร็จ เสริด เสริม แสร้ง โสรจ ไสร้ เฉพาะคำว่า “สระ” ออกเสียงได้ 3 อย่าง คือ อักษระเสียงพร้อมกัน 2 ตัวนับเป็นอักษรควบแท้ หรือ ถ้าออกเสียงแต่ตัว ส ไม่ออกเสียง ร นับเป็นอักษรควบไม่แท้ หรือออกเสียงคล้าย 2 พยางค์ นับเป็นอักษรนำ
2. ออกเสียงแปรไปเป็นเสียงอื่น ได้แก่ตัว ท ที่ควบกับตัว ร แล้วออกเสียงเป็นตัว ซ ดังตัวอย่างที่ผูกเป็นคำประพันธ์ไว้เพื่อจำง่าย ดังนี้ :-

ทรวดทรงทราบทรามทราย ทรุมโทรมหมายนกอินทรี
มัทรี อินทรีย์ มี เทริด นนทรี พุทรา เพรา
ทรวงไทยทรัพย์แทรกวัด โทรมนัสย์ ฉะเชิงเทรา
ตัว “ทฺร” เหล่านี้เรา ออกสำเนียงเป็นเสียง “ซ”

3. เป็นตัวสะกดด้วยกัน ได้แก่ตัว ร และ ห ที่ควบกับพยัญชนะตัวอื่น และเรียงไว้ข้างหน้าตัวอื่นที่มาควบด้วย แล้วทำหน้าที่เป็นตัวสะกดด้วยกัน แต่ให้พยัญชนะตัวหลังทำหน้าที่บังคับเสียงเป็นมาตราต่างๆ เช่น :-

ตรรก	อ่านว่า	ตัก	ร ควบ ก เป็นตัวสะกด
มรรค	”	มัก	ร ควบ ค เป็นตัวสะกด
อรรธ	”	อัด	ร ควบ ธ เป็นตัวสะกด
พรหม	”	พฺรม	ห ควบ ม เป็นตัวสะกด
พราหมณ์	”	พฺราม	ห ควบ ม เป็นตัวสะกด

4. การันต์ตัวหลัง ให้ตัวหน้าสะกดแต่ตัวเดียว ได้แก่ตัวอย่างในข้อ 3 นั่นเอง แต่แยกพยัญชนะที่ควบกับตัว ร ไปเป็นตัวการันต์ ให้ตัว ร ทำหน้าที่สะกดแต่ตัวเดียว เช่น :-

สรรพ์	อ่านว่า	สัน	ร สะกด พ การันต์
ครรภ์	อ่านว่า	คัน	ร สะกด ภ การันต์
หรรษ์	อ่านว่า	หัน	ร สะกด ษ การันต์

5.5.8.2 อักษรนำ

อักษรนำ คือพยัญชนะ 2 ตัวร่วมอยู่ในสระเดียวกัน บางคำก็ออกเสียงร่วมกันสนิท เช่น หนู หมอ อย่า อยาก , บางคำก็ออกเสียงคล้ายกับเป็น 2 พยางค์ เพราะพยัญชนะทั้ง 2 นั้นประสมกันไม่สนิทเหมือนอักษรควบแท้ จึงฟังคล้ายกับมีเสียงสระอะดังแผ่วออกมา เช่น :-

กนก	อ่านว่า	กฺนก	ไม่ใช่	กะ-หนก
ขนม	อ่านว่า	ขฺนม	ไม่ใช่	ขะ-หนม

โดยลักษณะของอักษรนำมีดังต่อไปนี้

1. ต้องเป็นพยัญชนะ 2 ตัวประสมกันร่วมอยู่ในสระตัวเดียวกัน
2. สระตัวใดที่ใช้เขียนไว้ข้างหน้า ก็ต้องเขียนไว้ข้างหน้าเช่น :- เขนย แสยง ไฉน , สระตัวใดใช้เขียนคร่อม ก็ต้องเขียนคร่อมอักษรนำเหมือนอักษรธรรมดา เช่น :- เฉลียว เขยื้อน , สระที่ใช้เขียนไว้ข้างบนข้างล่างและหลังพยัญชนะรวมทั้งรูปวรรณยุกต์ด้วย ให้เขียนไว้ที่ตัวที่ 2 เช่น :- สมี ฉลุย ถล่ม ขยุ้ม ฯลฯ
3. ตามปกติอักษรนำ เวลาออกเสียง จะปรากฏเสียงพยัญชนะ 2 ตัวประสมกันคนละครึ่ง พอสังเกตรู้ได้ว่าพยัญชนะอะไรประสมกัน แต่มียกเว้นพยัญชนะอยู่ 2 ตัวคือตัว ห กับตัว อ

   ตัว ห เมื่อเป็นตัวนำอักษรเดี่ยว ไม่ต้องออกเสียงคนละครึ่งเหมือนอักษรนำตัวอื่นๆ แต่ให้ออกเสียงประสมกันสนิทเหมือนอักษรควบแท้ เช่น :- หนู หงอ หมอ หยัน ใหญ่ ฯลฯ

   ตัว อ เมื่อนำหน้าตัว ย ว ก็ไม่ต้องออกเสียงเหมือนอักษรนำธรรมดา แต่ให้ออกเสียงทำนองเดียวกับตัว ห นำแต่เป็นเสียงอักษรกลาง เช่น :- อยู่ อย่า อย่าง อยาก และมีที่ใช้เพียง 4 ตัวนี้เท่านั้น

4. พยัญชนะตัวนำหน้าจะเป็นอักษรสูง อักษรกลาง หรืออักษรต่ำก็ได้ ส่วนพยัญชนะตัวหลังก็เช่นกัน แต่มีข้อสังเกตว่า ถ้าพยัญชนะตัวหน้าเป็นอักษรสูงหรืออักษรกลาง และพยัญชนะตัวหลังเป็นอักษรเดี่ยว อักษรเดี่ยวนั้นจะต้องออกเสียงและผันเสียงอย่างอักษรสูงหรือกลางซึ่งเป็นตัวนำ แต่ถ้าพยัญชนะตัวหลังมิใช่อักษรเดี่ยว ให้ออกเสียงอย่างปรกติ
5. จะใช้อักษรสูงนำอักษรสูง หรืออักษรต่ำนำอักษรต่ำก็ได้ แต่อักษรกลางนำอักษรกลางไม่มีที่ใช้
6. ถ้าเป็นตัวสะกดให้ถือตัวหน้าเป็นตัวสะกดแต่ตัวเดียว แต่ในเวลาเดียวกันก็ทำหน้าที่เป็นตัวนำด้วย เช่น :- วาสนา พิศวง ฯลฯ แต่ถ้าเป็นตัวการันต์จะเป็นทั้ง 2 ตัวหรือตัวเดียวก็ได้
7. วิธีสังเกตอักษรนำ

ก. สังเกตเสียง คือออกเสียงประสมกันแต่ไม่กล้ำกันสนิทเหมือนอักษรควบแท้ เว้นแต่ตัว ห และตัว อ เป็นตัวนำ

ข. สังเกตรูปสระ คือถ้าเป็นสระเอ แอ ไอ โอ จะต้องเขียนไว้ข้างหน้าอักษรนำทั้ง 2 ตัว ถ้าเป็นสระคร่อมต้องเขียนคร่อมทั้ง 2 ตัว

ค. สังเกตรูปพยัญชนะ คือถ้าพยัญชนะที่นำหน้าเป็นอักษรสูงหรือกลาง อักษรเดี่ยวซึ่งเป็นอักษรตัวที่ 2 จะต้องออกเสียงผิดปกติ

8. ตัว ห ใช้เป็นตัวนพเฉพาะอรรธสระ และพยัญชนะตัวสุดท้ายของพยัญชนะวรรคเท่านั้น เช่น :- หย่า หยก หยด หรอ หรูหรา หรี่ หลง หลน หล่น หวัง หวัด หวั่น หงอก หงอย หงาย หญ้า หญิง ใหญ่ หนู หนี หนวด หมอ หมด หมก
9. อักษรควบไม่แท้บางตัวก็ใช้เป็นอักษรนำได้ เช่น ปรารถนา

5.5.9 ตัวสะกด

ตัวสะกด คือพยัญชนะที่ประสมข้างหลังพยางค์ในแม่ ก กา แล้วทำหน้าที่บังคับแม่ ก กา ให้กลายเป็นแม่ กง กน กก กด กบ กม เกย หรือกล่าวอีกนัยหนึ่ง ได้แก่พยัญชนะที่ทำหน้าที่ประสมเป็นส่วนที่ 4 ของพยางค์ (คือ สระ + พยัญชนะ + วรรณยุกต์ + ตัวสะกด)

ตัวสะกดแบ่งออกเป็น 5 ชนิดคือ เป็นพยัญชนะโดด, เป็นอักษรควบแท้, เป็นอักษรควบไม่แท้, เป็นอักษรนำ, และเป็นนิคหิต

ตัวสะกดนั้นเมื่อประกอบข้างท้ายแม่ ก กา แล้ว มีอำนาจดังนี้คือ :- ทำให้สระเปลี่ยนรูปและลดรูป, ทำให้เสียงวรรณยุกต์เปลี่ยนไป เช่น นา (เสียงสามัญ) นาก (เสียงโท) เป็นต้น, ทำให้แม่ ก กา กลายเป็นแม่ต่างๆ ตามตัวสะกดที่บังคับเสียง, ทำให้ความหมายเปลี่ยนไป, ทำให้ลหุกลายเป็นครุ, และทำให้คำตายกลายเป็นคำเป็น และทำให้คำเป็นกลายเป็นคำตาย เช่น กิ-กิน มา-มาก เป็นต้น

5.5.10 ตัวหัน

ตัวหัน หรือ อักษรหัน คือพยัญชนะที่แปลงรูปวิสรรชนีย์มาเป็นพยัญชนะที่มีรูปเหมือนตัวสะกด แต่ทำหน้าที่แทนหันอากาศ และต้องเรียงไว้หน้าตัวสะกดที่มีรูปเหมือนกัน จึงเรียกว่าตัวหัน หรืออักษรหัน พยัญชนะทุกตัวที่เป็นตัวสะกดได้ย่อมเป็นตัวหันได้ทั้งสิ้น แต่บัดนี้นิยมใช้แต่ ร หันตัวเดียวนั้น

5.5.11 ตัวการันต์

ตัวการันต์ คือตัวอักษรซึ่งไม่ออกเสียง และมีไม้ทัณฑฆาตกำกับไว้เบื้องบน เพื่อแสดงว่า เสียงนั้นได้ถูกบังคับไว้จนกว่าจะมีพยางค์อื่นมาประสมข้างหลัง เช่น ทุกข์ สงฆ์ ทันต์ พันธุ์

ตัวการันต์นั้น จะมีรูปสระหรือไม่ก็ได้ถ้ามีจะต้องเป็นสระ อิ กับสระ อุ เท่านั้น ถ้าไม่มี จะต้องเป็นสระ อะ ลดรูป ตัวการันต์จำแนกออกได้เป็น 6 ชนิด คือ เป็นพยัญชนะตัวเดียว, เป็นพยัญชนะสองตัวเรียงกัน, เป็นพยัญชนะสามตัวเรียงกัน เช่น ลักษมณ์, เป็นอักษรควบแท้, เป็นอักษรควบไม่แท้, และเป็นอักษรนำ

5.6 พยางค์

สระ พยัญชนะ และวรรณยุกต์ ตามที่อธิบายมาข้างต้นนั้น เมื่อนำมาประสมกันแล้วทำให้เกิดเสียงเป็นพยางค์ต่างๆ

พยางค์ คือส่วนหนึ่งของคำหรือหน่วยเสียงที่ประกอบด้วยสระตัวเดียว จะมีความหมายหรือไม่มีก็ได้ พยางค์หนึ่งมีส่วนประสมต่างๆ กันคือ

1. พยัญชนะ + สระ + วรรณยุกต์ พยางค์ชนิดนี้เรียกว่าประสม 3 ส่วน
2. พยัญชนะ + สระ + วรรณยุกต์ + ตัวสะกด พยางค์ชนิดนี้เรียกว่าประสม 4 ส่วน
3. พยัญชนะ + สระ + วรรณยุกต์ + ตัวสะกด + การันต์ พยางค์ชนิดนี้เรียกว่าประสม 5 ส่วน

บทที่ 6
อัลกอริธึมที่ใช้ในการแยกพยางค์และแปลงข้อความไทยเป็นสัญลักษณ์การออกเสียง

6.1 อัลกอริธึมที่ใช้ในการแยกพยางค์

ในการแยกพยางค์ออกจากคำนั้น จำเป็นที่จะต้องรู้หลักต่างๆ ของภาษาดังที่กล่าวไปแล้ว ซึ่งเราสามารถนำหลักต่างๆ เหล่านั้นมาสรุปเป็นข้อๆ ได้ทั้งหมด 16 ข้อดังนี้

สัญลักษณ์	คำนิยาม
พ	พยัญชนะไทย
(พ)	พยัญชนะตัวควบกล้ำ ซึ่งอาจจะมีหรือไม่มีก็ได้
รร	ตัว ร หัน
ส นำ	สระนำในภาษาไทย
ส ตาม	สระตามในภาษาไทย
ด	ตัวสะกดที่ใช้ในภาษาไทย
[ย, อ]	พยัญชนะที่ใช้เป็นสระ ซึ่งจะขาดไม่ได้ เป็น ตัว ย หรือตัว อ เท่านั้น

ตารางที่ 6-1 ตารางสัญลักษณ์และคำนิยามที่ใช้ในการแยกพยางค์

กฎข้อที่ 1 การใช้อักษร อ นำ ย ซึ่งกรณีนี้เป็นกรณีพิเศษ
ได้แก่คำว่า อยาก อยู่ และคำพิเศษคือ ก็

กฎข้อที่ 2 การใช้อักษร อ นำ ย ซึ่งกรณีนี้เป็นกรณีพิเศษ
ได้แก่คำว่า อย่า อย่าง

กฎข้อที่ 3 การใช้ตัว ร หัน ในกรณีที่มีตัวสะกดตามหลัง สามารถแทนด้วยสัญลักษณ์ พ รร ด
ตัวอย่างเช่น พรรค วรรณ

กฎข้อที่ 4 การใช้ตัว ร หัน ในกรณีที่ไม่มีตัวสะกดตามหลัง สามารถแทนด้วยสัญลักษณ์ พ รร
ตัวอย่างเช่น พรรษา วรรณา

กฎข้อที่ 5 การใช้ สระ เอา เอาะ สามารถแทนด้วยสัญลักษณ์ เ พ พ า (ะ)
ตัวอย่างเช่น เพราะ เพลา

กฎข้อที่ 6 การใช้ คำที่อ่านออกเสียงเป็น เอิง สามารถแทนด้วยสัญลักษณ์ เ พ (พ) ด
ตัวอย่างเช่น เพลิง เทิง

กฎข้อที่ 7 การใช้ สระ เออะ เออ สามารถแทนได้ด้วยสัญลักษณ์ เ พ อ (ะ)

ตัวอย่างเช่น เลอะเทอะ เผลอ เจอ

กฎข้อที่ 8 การใช้ สระ เอียะ เอีย เอือะ เอือ ตามด้วยตัวสะกด สามารถแทนได้ด้วยสัญลักษณ์ เ พ (พี, พี) [ย, อ] ด

ตัวอย่างเช่น เพื่อน เรียน เกลียด

กฎข้อที่ 9 กฎของการใช้ สระ เอียะ เอีย เอือะ เอือ สามารถแทนได้ด้วยสัญลักษณ์ เ พ (พี, พี) [ย, อ]

ตัวอย่างเช่น เสีย เผียะ เพื่อ เกือะ

กฎข้อที่ 10 การใช้ สระนำ โอะ โอ ไอ ใอ เอะ เอ สามารถแทนด้วยสัญลักษณ์ ส นำ พ พ (ะ)

ตัวอย่างเช่น ไกล โกน โต๊ะ ใจ เกะ เห

กฎข้อที่ 11 การใช้ สระ โอ ไอ ใอ เอ ที่มีตัวสะกดสามารถแทนด้วยสัญลักษณ์ ส นำ พ (พ) ด

ตัวอย่างเช่น โอ่ง โล่ง โกน เพลง

กฎข้อที่ 12 การใช้ พยัญชนะแล้วตามด้วยสระ ตัวสะกดสามารถแทนด้วยสัญลักษณ์ พ (พ) ส ตาม ด

ตัวอย่างเช่น กราม พราหม กัด

กฎข้อที่ 13 การใช้ พยัญชนะ สระ สามารถแทนด้วยสัญลักษณ์ พ (พ) ส ตาม

ตัวอย่างเช่น พระ กะทิ

กฎข้อที่ 14 การใช้ พยัญชนะ ตัวสะกด สามรถแทนด้วยสัญลักษณ์ พ (พ) ด

ตัวอย่างเช่น กด ห่ม

กฎข้อที่ 15 การใช้ พยัญชนะ ที่ตามด้วยตัวสะกด ร ล สามารถแทนด้วยสัญลักษณ์ พ [ร, ล]

ตัวอย่างเช่น พร พล กร กล

กฎข้อที่ 16 การใช้ พยัญชนะ ที่ออกเสียง อะ กึ่งเสียง สามารถแทนด้วยสัญลักษณ์ พ

ตัวอย่างเช่น ขจี อนัตตา สนิท

6.2 อัลกอริธึมที่ใช้ในการแปลงเป็นสัญลักษณ์การออกเสียง

ในการแปลงข้อความไทยเป็นสัญลักษณ์การออกเสียง (phonetic symbols) นั้นจะเริ่มจาก เมื่อเราตัดคำออกมาจากประโยคแล้ว นำคำที่ได้จากขั้นตอนการตัดคำนั้นมาแยกพยางค์ โดยใช้หลักภาษาไทยทั้ง 16 ข้อที่ได้กล่าวมาแล้ว จากนั้นจึงนำคำที่แยกพยางค์แล้วมาแปลงเป็นสัญลักษณ์การออกเสียง โดยวิธีการแปลงนั้นจะเริ่มจากนำพยางค์ที่ได้มาแยกเอาพยัญชนะต้นออกมาก่อน ซึ่งในขั้นตอนนี้จำเป็นต้องระบุด้วยว่าเป็นพยัญชนะในกฎข้อใดของไตรยางค์เพื่อจะได้นำมาใช้ในการผันเสียง จากนั้นจึงเริ่มแยกสระออกมาซึ่งตรงนี้ก็ยังต้องระบุอีกว่าเป็นสระเสียงสั้นหรือเสียงยาว และในกรณีที่ไม่มีตัวสะกดจะทำให้เป็นคำเป็นหรือคำตาย ต่อจากนั้นจึง

ตรวจดูว่าพยางค์ที่นำเข้ามามีตัวสะกดหรือไม่ และเช็คว่าเป็นตัวสะกดคำเป็นหรือคำตาย สุดท้ายจึงเริ่มผันเสียงวรรณยุกต์

ในส่วนของสัญลักษณ์การออกเสียงนั้น จะมีรูปแบบการแสดงผลดังนี้คือ /พยัญชนะต้น_สระ_ตัวสะกด_เสียงวรรณยุกต์/ ส่วนสัญลักษณ์ของการออกเสียงนั้น จะมีอยู่ในตารางดังต่อไปนี้

อักษรไทย	สัญลักษณ์การออกเสียง	
	พยัญชนะต้น	ตัวสะกด
ก	/k/	/k/
ข , ฃ , ค , ฅ , ฆ	/kh/	/k/
ง	/ng/	/ng/
จ	/c/	/t/
ฉ , ช , ฌ	/ch/	/t/
ซ , ศ , ษ , ส	/s/	/t/
ญ , ย	/j/	/j/ ตัว ย , /n/ ตัว ญ
ฎ , ด	/d/	/t/
ฏ , ต	/t/	/t/
ฐ , ฑ , ฒ , ถ , ท , ธ	/th/	/t/
ณ , น	/n/	/n/
บ	/b/	/p/
ป	/p/	/p/
พ , ภ , ผ	/ph/	/p/
ฟ , ฝ	/f/	/p/
ม	/m/	/m/
ร	/r/	/n/
ล , ฬ	/l/	/n/
ว	/w/	/w/
ห , ฮ	/h/	-
อ	/?/	-

ตารางที่ 6-2 ตารางสัญลักษณ์การออกเสียงพยัญชนะต้นและตัวสะกด

เสียงสระ	สัญลักษณ์การออกเสียง
อะ	/a/
อา	/a;/
อิ	/i/
อี	/i;/
อึ	/v/
อื	/v;/
อุ	/u/
อู	/u;/
เอะ	/e/
เอ	/e;/
แอะ	/x/
แอ	/x;/
โอะ	/o/
โอ	/o;/
เอาะ	/@/
ออ	/@;/
เออะ	/#/
เออ	/#;/
เอียะ	/ia/
เอีย	/i;a/
เอือะ	/va/
เอือ	/v;a/
อัวะ	/ua/
อัว	/u;a/
อำ	/am/
ไอ , ใอ	/aj/
เอา	/aw/

ตารางที่ 6-3 ตารางแสดงสัญลักษณ์การออกเสียงสระ

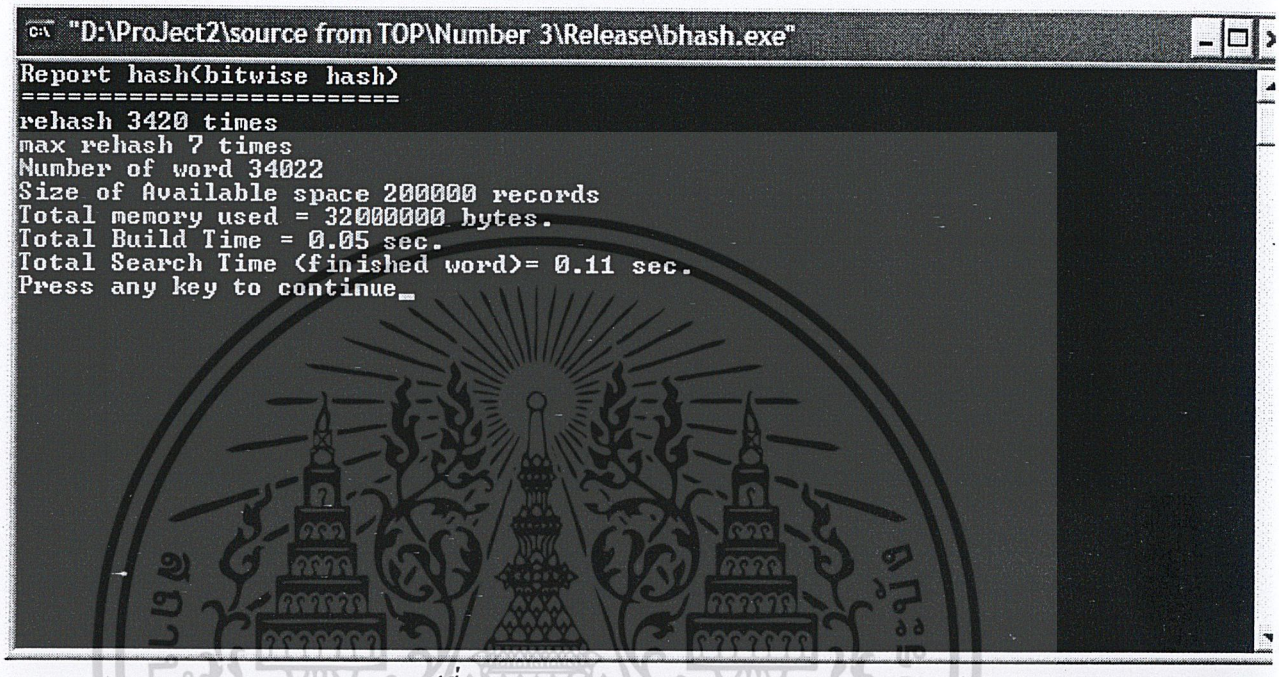
เสียงวรรณยุกต์	สัญลักษณ์การออกเสียง
สามัญ	/0/
เอก	/1/
โท	/2/
ตรี	/3/
จัตวา	/4/

ตารางที่ 6-4 ตารางแสดงสัญลักษณ์การออกเสียงวรรณยุกต์

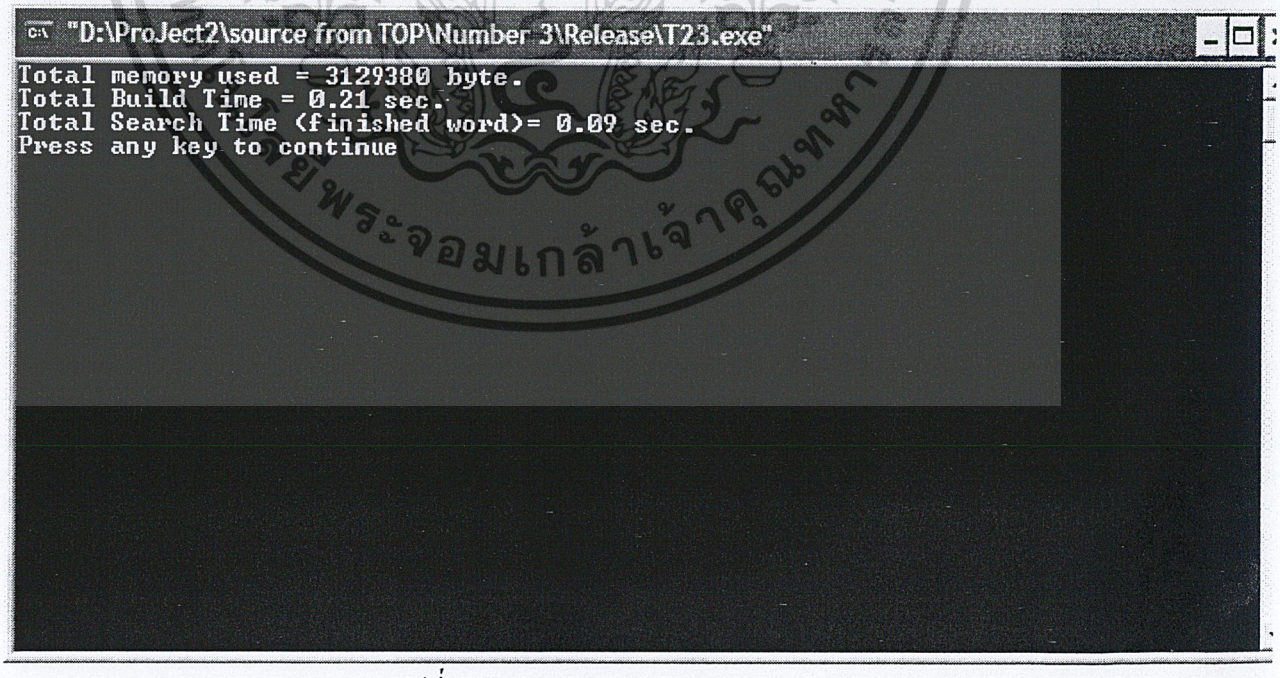
บทที่ 7

ผลการทดลอง บทสรุปและวิจารณ์

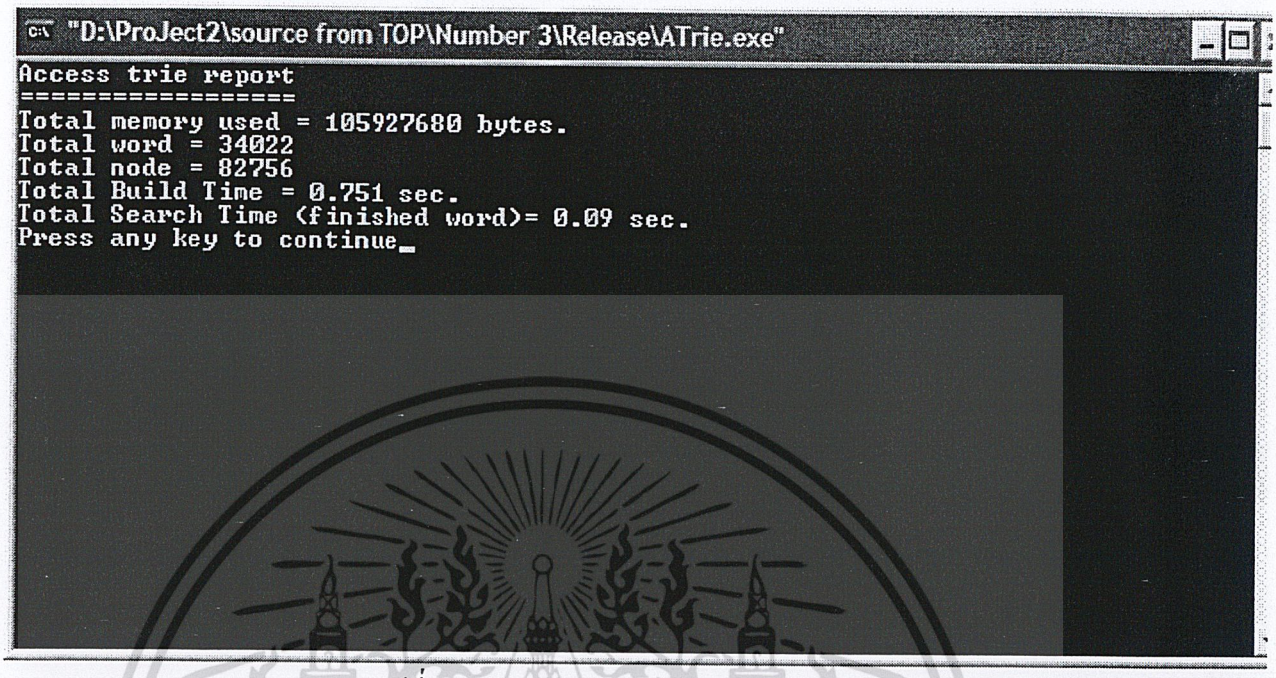
7.1 ผลการทดลอง



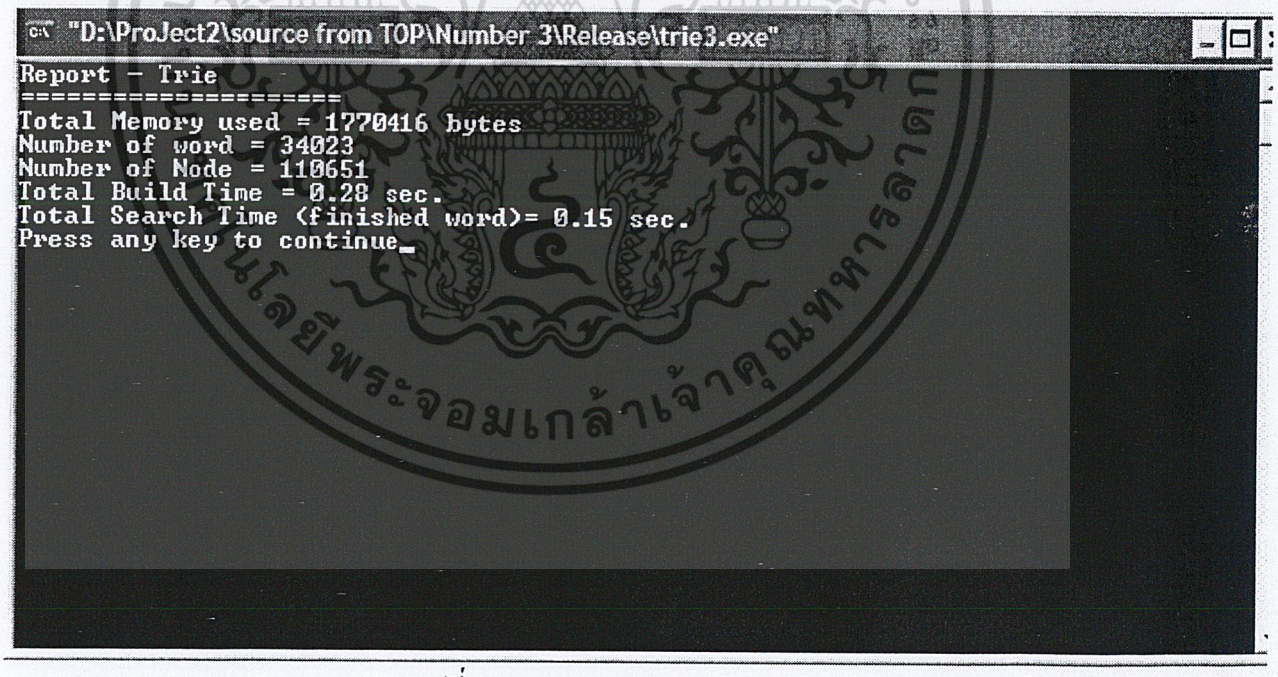
รูปที่ 7-1 ผลการทดลอง hash



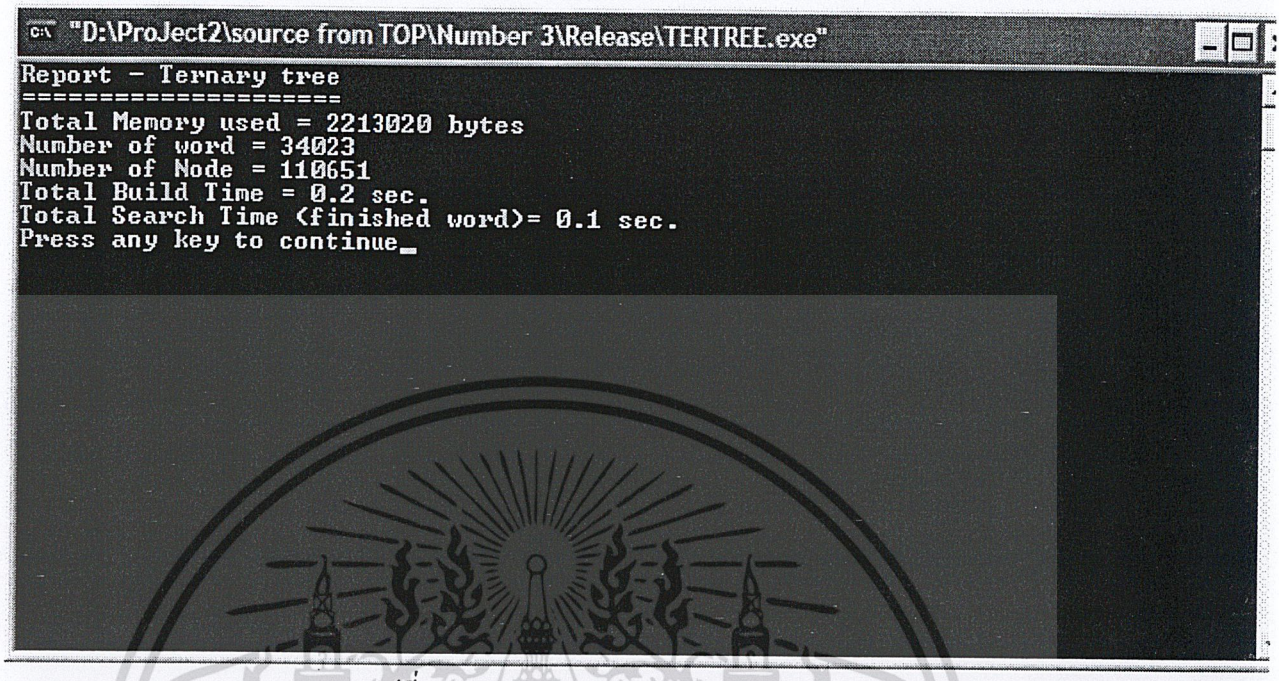
รูปที่ 7-2 ผลการทดลอง 2-3 tree



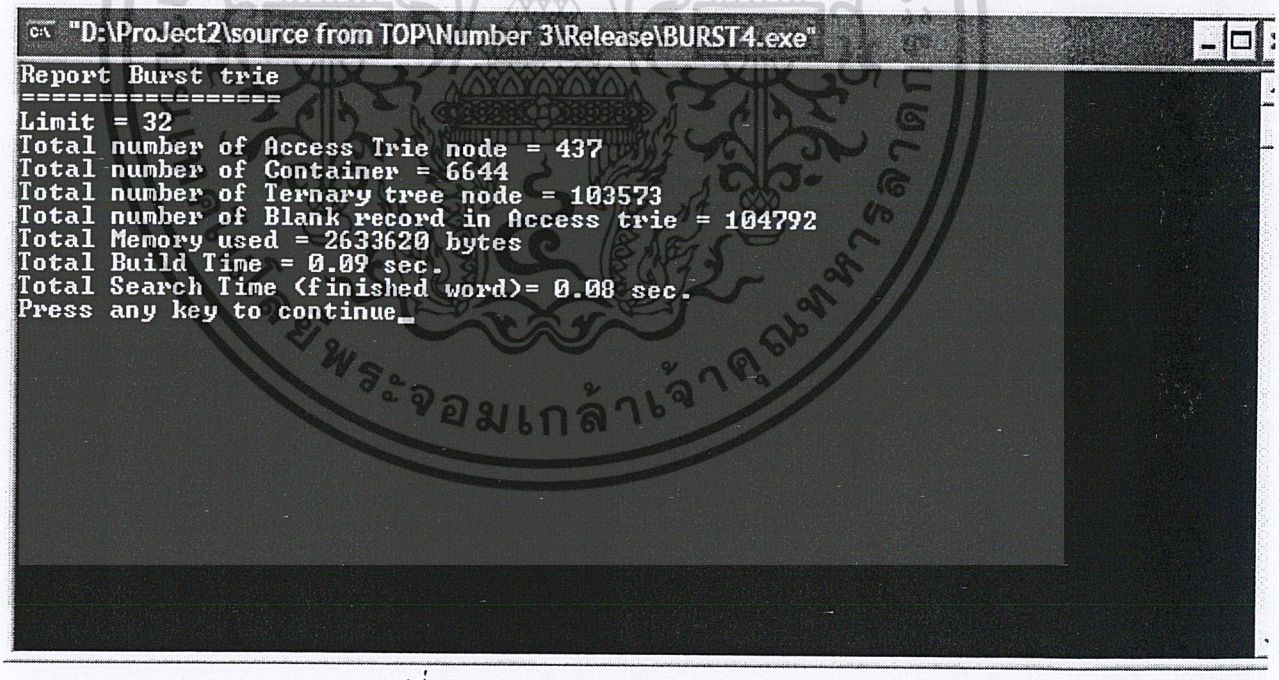
รูปที่ 7-3 ผลการทดลอง access trie



รูปที่ 7-4 ผลการทดลอง trie



รูปที่ 7-5 ผลการทดลอง ternary tree



รูปที่ 7-6 ผลการทดลอง Burst Trie

7.2 สรุปและวิจารณ์

1. โครงสร้างข้อมูลที่เหมาะสมที่สุดในการนำมาประยุกต์ใช้เป็นพจนานุกรมสำหรับการตัดคำคือ โครงสร้างข้อมูลแบบ Ternary tree เพราะมีคุณสมบัติตรงตามความต้องการทุกประการคือ มีความรวดเร็วในการทำงาน ใช้เนื้อที่หน่วยความจำอย่างมีประสิทธิภาพ สามารถรับ input ทีละตัวอักษรได้ และนำไปประยุกต์ใช้ได้ง่าย

2. การตัดคำทั้งแบบ 1 parse และ 2 parse เป็นการตัดคำที่ทำงานได้อย่างรวดเร็ว แต่การตัดคำแบบ 1 parse มีข้อดีคือ สามารถตัดคำได้แม้ว่ายังอ่าน input ไม่จบ จึงเหมาะกับงานที่มี input เข้ามาเรื่อยๆ แต่ก็มีข้อเสียคือ ไม่สามารถให้คำตอบได้ทุกกรณี ซึ่งในการตัดคำแบบ 2 parse สามารถให้คำตอบได้ทุกกรณี แต่จำเป็นต้องอ่าน input ให้จบเสียก่อนจึงจะเริ่มทำการตัดคำได้

3. ในการแยกพยางค์นั้น มีความถูกต้องในการแยกพยางค์ถึง 95 % ซึ่งในส่วนนี้จะมีส่วนที่การแยกพยางค์ผิดพลาดในส่วนตรงที่ตัวสะกดเป็นลักษณะอักษรประสมเช่น สามารถ จะแยกได้เป็น สา-มา-รถ และในลักษณะคำที่มาจากภาษาบาลีสันสกฤตซึ่งมักจะมีการออกเสียง อะ กึ่งเสียงอยู่ จะไม่สามารถแยกพยางค์ออกมาในรูปนั้นได้

4. ในส่วนของการแปลงเป็นสัญลักษณ์การอ่านนั้นสามารถทำได้ถูกต้อง โดยเมื่อได้รับคำมาจากฟังก์ชั่นการแยกพยางค์แล้วสามารถแปลงได้ตรงที่ได้รับมา โดยจะยังมีส่วนที่ยังไม่ได้ทำคือ การแปลงเสียงวรรณยุกต์ให้ถูก โดยในขณะนี้ทำไว้ให้เปลี่ยนสัญลักษณ์การออกเสียงให้ตรงกับรูปของวรรณยุกต์เท่านั้น

7.3 ข้อเสนอแนะ

โปรแกรมตัดคำนี้ ยังสามารถเพิ่มความเร็วในการทำงานได้อีก โดยเปลี่ยนโครงสร้างข้อมูลที่ใช้ จาก ternary tree เป็น burst trie ซึ่งมีความรวดเร็วในการค้นหาคำมากกว่า และในส่วนของการแยกพยางค์ ควรจะมีการนำเอา A.I. เข้าใช้เพื่อช่วยให้สามารถแยกพยางค์ได้ถูกต้อง โดยนำ A.I. เข้ามาช่วยดูบริบทข้างเคียงเพื่อตัดสินใจว่าคำนั้นควรแยกพยางค์ออกมาอย่างไร และส่วนของการแปลงเป็นสัญลักษณ์การออกเสียงนั้น ควรใช้วิธีเปลี่ยนสัญลักษณ์การออกเสียงจากรูปวรรณยุกต์มาเป็นตามเสียงที่ถูกต้องของพยางค์แทน ซึ่งจำเป็นต้องมีการศึกษาเรื่องการผันวรรณยุกต์ให้ลึกซึ้งขึ้น

บรรณานุกรม

หนังสืออ้างอิง

(1) กำชัย ทองหล่อ : “หลักภาษาไทย”, รวมสาส์น (1977), กรุงเทพฯ , 2543

เอกสารอ้างอิง

(1) Justin Zobel, Steffen Heinz, Hugh E. Williams : “In-memory Hash Tables for Accumulating Text Vocabularies”, Department of Computer Science, RMIT University

(2) M.V. Ramakrishna, Justin Zobel : “Performance in Practice of String Hashing Functions”, Department of Computer Science, RMIT University

(3) Steffen Heinz, Justin Zobel : “Performance of Data Structures for Small Sets of Strings”, School of Computer Science and Information Technology, RMIT University

(4) Steffen Heinz, Justin Zobel, Hugh E. Williams : “Burst Tries: A Fast, E_cient Data Structure for String Keys” School of Computer Science and Information Technology, RMIT University

(5) “ข้อเสนอการสร้างฐานข้อมูลสำหรับ LVCSR ภาษาไทย”