บทที่ 1

บทนำ

ความสำคัญ และความเป็นมาของปัญหา

ระบบจัดการฐานข้อมูลพจนานุกรมที่มีอยู่ในปัจจุบันเป็นเพียงการเก็บคำศัพท์ต่างๆเป็นคีย์เพื่อใช้ในการตรวจสอบคำ หรือแบ่งคำเป็นส่วนใหญ่ จึงยังขาดความเร็วและความยืดหยุ่นในการเพิ่มลบข้อมูล แต่ฐานข้อมูลพจนานุกรมภาษาไทยที่จะสร้างขึ้นนี้เป็นคลังศัพท์ขนาดใหญ่ หากไม่มีความสามารถในการเพิ่มลบคำศัพท์แล้ว ในการเพิ่มลบคำศัพท์แต่ละครั้งจะต้องเสียเวลาในการสร้างพจนานุกรมทั้งหมดขึ้นมาใหม่ จึงควรที่จะสร้างโครงสร้างข้อมูลระบบจัดการฐานข้อมูลพจนานุกรมภาษาไทยที่มีทั้งประสิทธิภาพในเรื่องเนื้อที่ที่ใช้เก็บพจนานุกรม และความเร็วในการสืบค้นเพิ่มลบคำเพื่อลบจุดบกพร่องของพจนานุกรมดังกล่าว และเมื่อพัฒนาระบบจัดการฐานข้อมูลด้วยโปรแกรมภาษาซีเสร็จ สามารถนำไปใช้เป็นโครงสร้างข้อมูลของพจนานุกรมมาตรฐานสำหรับโปรแกรมประยุกต์อื่นๆได้

พจนานุกรมภาษาไทยที่นำมาสร้างเป็นฐานข้อมูลเริ่มต้นจากงานวิจัยของ ยืน ภู่วรวรรณ [1] ซึ่งสร้างขึ้นมาใช้ในการแบ่งคำโดยพัฒนาบนระบบดอสเป็นเพียงงานทดลองเพื่อทดสอบกับระบบแบ่งคำที่ใช้กฎ จึงด้อยประสิทธิภาพทั้งในเรื่องของขนาดหน่วยความจำและความเร็ว ถัดมาคือผลงานวิจัยของ ควงแก้ว สวามิภักดิ์ [2] โดยใช้โครงสร้างข้อมูล B-Tree ผ่านทางเครื่องมือซอฟต์แวร์ (Software Tool) ที่เป็นระบบจัดการฐานข้อมูล งานทั้งหมดพัฒนาบนระบบปฏิบัติการยูนิกซ์ และมีจุดประสงค์เพื่อใช้ในการแบ่งคำ ผลงานวิจัยนี้ใช้คำศัพท์ในฐานข้อมูลมาเทียบกับประโยคที่จะแบ่งคำ ต่อมาเป็นงานโครงงานของนิสิตวิศวกรรม คอมพิวเตอร์ สัมพันธ์ ระรื่นรมย์ [3] ที่พัฒนาจากโครงสร้างทรัยอัดแน่น (Packed Trie) ของ Liang [4] และ Fredkin [5] โครงสร้างพจนานุกรมข้างต้นล้วนมีลักษณะสถิตย์ (Static) ยังขาดประสิทธิภาพในการเพิ่มหรือลบคำในฐานข้อมูล แต่เมื่อเทียบกับการใช้โครงสร้างทรัยแถวคู่ของ Aoe [6][7][8][9] ซึ่งมีลักษณะจลน์ (Dynamic) สามารถแก้ไขจุดบกพร่องนี้ได้

นอกจากงานวิจัยทางการศึกษาแล้ว ยังมีงานเชิงพาณิชย์เป็นโปรแกรมใช้งานจัดพิมพ์เอกสารซึ่งสร้างพจนานุกรมมาใช้ในการแบ่งคำอีกสองสามชิ้น พบว่างานส่วนใหญ่ที่สร้างขึ้นไม่ได้สร้างเพื่อที่

จะใช้ประโยชน์เป็นพจนานุกรมที่สามารถอ้างอิงความหมายหรือส่วนอื่นๆของคำศัพท์ได้ จึงขาดประสิทธิภาพในส่วนเพิ่มลบคำศัพท์

1. โครงสร้างข้อมูล

มีโครงสร้างข้อมูลหลายแบบที่สามารถใช้ในการแทนพจนานุกรม โดยแต่ละแบบจะมีความต่างกันในเรื่องของขนาดและความเร็วในการค้นหา ในที่นี้จะไม่ลงไปในรายละเอียดของแต่ละโครงสร้าง แต่จะพูดถึงข้อดีข้อเสียอย่างคร่าวๆ จนกว่าจะถึงโครงสร้างที่เลือกใช้คือทรัยแถวคู่

โครงสร้างเชิงรายการแบบลำดับ (Sequential List) เป็นวิธีที่เก็บคำอย่างตรงไปตรงมาที่สุด มีขนาดและใช้เวลาเป็นสัดส่วนกับจำนวนคำในพจนานุกรม โครงสร้างเชิงรายการแบบลำดับ (Sequential List) เป็นวิธีที่เราใช้เก็บกลุ่มคำก่อนที่จะนำมาเก็บโดยใช้วิธีอื่น และถือเป็นขนาดอ้างอิงสำหรับอัตราส่วนการลดขนาด (Compression Ratio)

โครงสร้างเชิงรายการแบบเรียงลำดับ (Sorted List) เก็บคำตามลำดับของคีย์ เมื่อใช้การค้นแบบทวิภาค (Binary Search) จะใช้เวลาเป็นล็อกการิธึมของขนาดพจนานุกรม แต่ขนาดไม่เปลี่ยนแปลงหรืออาจจะเพิ่มขึ้นสำหรับการประยุกต์ใช้ตารางตัวชี้

โครงสร้างเชิงรายการแบบดรรชนี (Indexed List) เพิ่มความเร็วให้กับโครงสร้างเชิงรายการแบบลำดับ (Sequential List) (ซึ่งต้องจัดลำดับด้วยถ้าจะใช้ในการแบ่งคำ) มีผลต่อขนาดขึ้นอยู่กับวิธีการทำดรรชนี

โครงสร้างต้นไม้แบบค้นทวิภาค (Binary Search Tree) ใช้เวลาเป็นล็อกการิธึม อาจจะคิดว่าเป็นอีกรูปแบบหนึ่งของโครงสร้างเชิงรายการแบบเรียงลำดับ (Sorted List) ก็ได้ ตัวอย่างเช่นถ้าหากทราบความน่าจะเป็นของการค้นหาคีย์ในต้นไม้ ก็สามารถปรับต้นไม้ เพื่อให้ค้นหาเร็วขึ้นได้ วิธีนี้ให้ความสะดวกในการเพิ่มลบ แต่มีปัญหาเรื่องลำดับการเพิ่มคีย์ในบางกรณีซึ่งจะทำให้ใช้เนื้อที่แย่มาก

ต้นไม้สมดุล (Balanced Tree) (รวมต้นไม้เอวีแอล, ต้นไม้ 2-3 และต้นไม้แบบบี) เช่นเดียวกับโครงสร้างต้นไม้แบบค้นทวิภาค (Binary Search Tree) แต่ตัดปัญหาเรื่องการลำดับการเพิ่มคีย์ เพราะมีการสมดุลตัวเองตลอดเวลา ต้นไม้แบบบีเหมาะสำหรับโปรแกรมขนาดใหญ่ เพราะถูกออกแบบมาให้เข้าถึงหน่วยความจำสำรองน้อยที่สุดในการค้นหาคีย์ อย่างไรก็ตามไม่ประหยัดเนื้อที่เพราะต้องใช้ตัวชี้เพิ่ม

การหาเลขที่อยู่แบบแฮช (Hashing) เป็นวิธีที่เพิ่มความเร็วในการค้นหาขึ้นจากโครงสร้างเชิงรายการแบบลำดับ (Sequential List) มาก และในกรณีทั่วไปน่าจะเร็วกว่าโครงสร้างต้นไม้แบบค้นทวิภาค (Binary Search Tree) โดยคำนวณตำแหน่งที่เก็บคีย์ที่ค้นหาจากฟังก์ชั่นเฉพาะอันหนึ่ง แต่ไม่มีข้อได้เปรียบเรื่องขนาด

การลงรหัสแบบเขียนทับ (Superimposed Coding) เป็นแบบหนึ่งของการหาเลขที่อยู่แบบแฮช (Hashing) ซึ่งจะไม่เก็บตัวคำเหลือแต่เพียงแถวลำดับของบิต ทำให้ลดขนาดของพจนานุกรมลงได้อย่างมาก โดยยอมให้มีความผิดพลาดได้เล็กน้อย

2. โครงสร้างทรัย

ความแตกต่างของทรัยมาจากการมองลักษณะของข้อมูลเป็นลำดับจากเซ็ตของตัวอักษรโดยตรง แทนที่จะมองแต่ละคำเป็นข้อมูลชิ้นหนึ่ง ทำให้จัดการกับข้อมูลขนาดแปรเปลี่ยนได้ดี และเข้ากันได้ดีกับอัลกอริธึมที่ทำงานกับสตริงทีละตัวอักษรเช่น การเปรียบแบบ (Pattern Matching) หรืออัลกอริธึมแบ่งคำ

ข้อดีของทรัยมาจากการจัดการกับอุปสรรค (prefix) ร่วมของคำในพจนานุกรมโดยการยุบรวมกันหมด ทำให้ลดขนาดพจนานุกรมลงได้ในทันที ขณะที่อำนวยความสะดวกในเรื่องการค้นหาไปด้วยพร้อมกัน

ทรัยเป็นต้นไม้แบบหลายทาง (m-ary tree) โดย m เป็นจำนวนตัวอักษรในเซ็ตของตัวอักษร แต่ละบัพ (node) ของทรัยคือสถานะ (state) ในอัตโนมัติจำกัด (Finite Automation) และแต่ละเส้นเชื่อม (edge) ซึ่งมีตัวอักษรหนึ่งตัวกำกับอยู่คือการผ่าน (transition) ในอัตโนมัติจำกัด (Finite Automation) มีหลายวิธีที่จะแทนทรัยในหน่วยความจำขึ้นอยู่กับวิธีการเก็บบัพและเส้นเชื่อมเหล่านี้

เราจะใช้เซ็ตของคำศัพท์ภาษาอังกฤษที่ใช้บ่อย 31 คำในรูปที่ 1.1 แสดงทรัยแบบต่างๆ

A	FOR	IN	THE
AND	FROM	IS	THIS
ARE	HAD	IT	TO
AS	HAVE	NOT	WAS
AT	HE	OF	WHICH
BE	HER	ON	WITH
BUT	HIS	OR	YOU
BY	I	THAT	

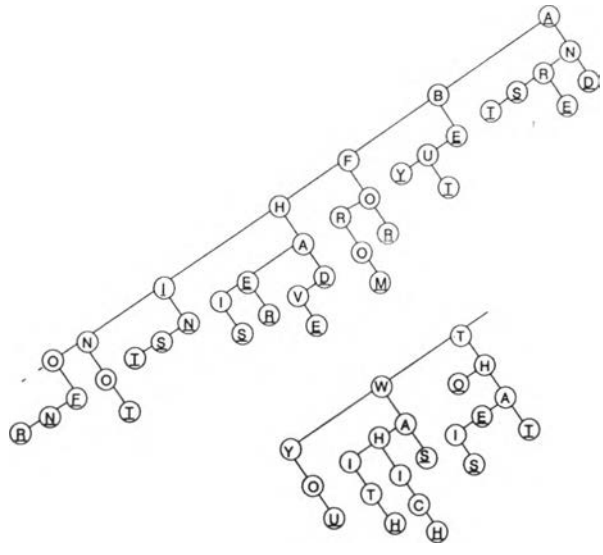
รูปที่ 1.1 คำภาษาอังกฤษที่ใช้บ่อย 31 คำ

2.1 ทรัยโยง (Linked Trie)

ทรัยโยง คือการแปลงทรัยซึ่งเป็นต้นไม้แบบหลายทาง (m-ary tree) ให้เป็นต้นไม้แบบทวิภาค (binary tree) (ดูรูปที่ 1.2) แต่ละบัพแสดงถึงการตรวจสอบกับลำดับของตัวอักษรของคำที่ค้นหา หากตัวอักษรที่นำมาค้นหาตรงกับในบัพ ก็จะนำตัวอักษรถัดไปในคำที่ค้นหามาเปรียบเทียบกับบัพในกิ่งด้านขวา และหากตัวอักษรในบัพขีดเส้นใต้ไว้แสดงว่าเป็นบัพสุดท้าย นั่นคือคำที่ค้นหามีอยู่ในเซ็ต และหากตัวอักษรนั้นไม่ตรงกับในบัพ ก็ไปตรวจสอบกับตัวอักษรในบัพทางกิ่งด้านซ้าย จะเห็นว่าไม่จำเป็นต้องเก็บทั้งคีย์ลงในทรัย

อย่างไรก็ดีทรัยโยงยังค่อนข้างช้าอยู่เนื่องจากการตรวจสอบกับแต่ละตัวอักษรของคำยังเป็นแบบเรียงลำดับ ซึ่งจำนวนการเปรียบเทียบที่มากที่สุดคือ m หรือจำนวนตัวอักษรในเซ็ตของตัวอักษร

แต่ทรัยโยงนั้นง่ายต่อการเพิ่มลบคีย์ แต่กินเนื้อที่มากเกินจำเป็นเนื่องจากมีตัวชี้ถึงสองอันต่อหนึ่งเส้นเชื่อม ถ้าตัดเรื่องการเพิ่มลบคีย์ทิ้งไป เราสามารถลดตัวชี้ได้หนึ่งอันคือตัวที่ชี้ไปยังกิ่งด้านขวา โดยการเก็บเส้นเชื่อมของบัพเดียวกันไว้ติดกัน เรียกการเก็บแบบนี้ว่าทรัยโยงแบบหลายทาง (Multi-way Linked Trie)



รูปที่ 1.2 ทรัยโยงของคำอังกฤษที่ใช้บ่อย

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	2	5				7		11	16					17	19					20			24		31	
2														3				4	0	0						
3				0																						
4					0																					
5					0																6				0	
6																				0						
7															8			9								
8																		0								
9															10											
10													0													
11	12				14				15																	
12				0																		13				
13					0																					
14																		0								
15																			0							
16														0					0	0						
17															18											
18																				0						
19						0								0				0								
20								21							0											
21	22				0				23																	
22																				0						
23																			0							
24	25							26	29																	
25																			0							
26									27																	
27			28																							
28								0																		
29																				30						
30								0																		
31															32											
32																					0					

รูปที่ 1.3 ทรัยครรชนีของคำอังกฤษที่ใช้บ่อย

2.2 ทรัยดรรชนี (Indexed Trie)

ทรัยโยงมีข้อเสียจากการที่ต้องทำการค้นหาเส้นเชื่อมในบัพอย่างเรียงลำดับ ในทางตรงกันข้ามทรัยดรรชนีจะอาศัยค่าของตัวอักษรในการทำดรรชนีเข้าไปในบัพเลยโดยไม่ต้องค้นหาเส้นเชื่อม

ทรัยดรรชนีจะเก็บแต่ละบัพเป็นอาร์เรย์ขนาดคงที่ขนาดเท่าจำนวนตัวอักษรในเซ็ตของตัวอักษร โดยมีเส้นเชื่อมเป็นสมาชิกของอาร์เรย์ แต่ละเส้นเชื่อมจะมีเพียงตัวชี้ไปยังบัพต่อไปและตัวบ่งชี้ (flag) เท่านั้นโดยไม่ต้องเก็บตัวอักษร แต่จะอาศัยการเก็บแต่ละเส้นเชื่อมไว้ที่ตำแหน่งจากอาร์เรย์เท่ากับค่าของตัวอักษรนั้น (เช่นให้ A=0, B=1 ...) และตำแหน่งใดที่ไม่มีเส้นเชื่อมจะเก็บค่าตัวชี้ว่าง (NULL pointer) เพื่อบอกว่าช่องนี้ใช้ไม่ได้ (invalid) (ดูรูปที่ 1.3)

ทรัยดรรชนีเป็นอาร์เรย์ 2 มิติที่มีสดมภ์เป็นเส้นเชื่อมและมีแถวเป็นบัพ โดยที่ฐาน (base) ของแต่ละบัพอยู่ที่ตำแหน่งศูนย์จากแต่ละแถว

ในแง่ความเร็ว ทรัยดรรชนีเป็นวิธีที่เร็วมาก การค้นหาคำในทรัยดรรชนีจะใช้เวลาเท่ากับการอ่านตัวอักษรแต่ละตัวจากอินพุตเท่านั้น (O(n)) เนื่องจากแต่ละขั้นตอนเป็นเพียงการทำดรรชนีในอาร์เรย์ นอกจากนั้นทรัยดรรชนียังสะดวกต่อการเพิ่มลบคีย์ แต่ปัญหาคือทรัยดรรชนีใช้เนื้อที่อย่างไม่ประหยัดโดยสิ้นเชิง เพราะสมาชิกส่วนใหญ่ในอาร์เรย์ของแต่ละบัพจะว่าง มีเพียงบัพราก (Root Node) เท่านั้นที่น่าจะที่มีเส้นเชื่อมอยู่เต็ม

2.3 ทรัยอัดแน่น (Packed Trie)

ทรัยอัดแน่นเป็นโครงสร้างข้อมูลที่เสนอโดย Liang [4] รวมเอาข้อดีของทั้งทรัยโยงและทรัยดรรชนีไว้ด้วยกันทั้งเรื่องขนาดและความเร็ว โดยอาศัยแนวความคิดเกี่ยวกับการจัดการตารางแบบไม่แน่น (Sparse Table) หรือการอัดตารางกระจาย (Parsing Table)

	0	1	2	3	4	5	6	7	8	9
00		A 8	B 11			D 0	F 3	E 0	H 30	I 23
10	C 5			H 0	N 25	O 32	E 0		O 12	M 0
20	T 33	R 14	N 1	W 46	T 0	Y 37	R 2	S 0	T 0	O 6
30	R 0	A 29	U 4	D 0	S 0	E 12	Y 0	N 0	F 0	I 15
40	O 4	H 44	S 0	T 0	I 7	A 4	N 0	A 15	O 0	E 0
50	R 0	V 2	O 38	I 15	H 35	I 36	T 5			U 0

รูปที่ 1.4 ทรัยอัดแน่นของคำอังกฤษที่ใช้บ่อย

หลักการง่ายๆ คือการใช้เนื้อที่ที่เคยเป็นตัวชี้ว่างในทรัยดรรชนีให้เป็นประโยชน์ วิธีการคือยอมเก็บตัวอักษรของเส้นเชื่อมไว้กับเส้นเชื่อมแต่ละอัน และมองทรัยอัดแน่นเป็นอาร์เรย์ 1 มิติ ขนาดใหญ่อันเดียว คือยอมให้ฐานของแต่ละบัพอยู่ที่ตำแหน่งเท่าไรก็ได้ แต่ต้องไม่ซ้ำกับฐานของบัพอื่น

การทำดรรชนีแต่ละครั้งจะนับไปจากฐานของบัพปัจจุบัน และต้องตรวจดูก่อนว่าตัวอักษรที่ตำแหน่งนั้นตรงกับที่ใช้เป็นดรรชนีหรือไม่ ถ้าไม่แสดงว่าจริงๆแล้ว ตัวชี้ที่ตำแหน่งนั้นเป็นว่าง แต่ช่องนั้นถูกใช้โดยบัพอื่นเพื่อเก็บเส้นเชื่อมของบัพนั้น ไม่มีทางที่จะบังเอิญมีการเข้าใจผิด เนื่องจากการบังคับห้ามบัพต่างๆใช้ฐานเดียวกัน แม้ว่าการใช้เนื้อที่ของทั้งสองจะเข้ากันได้พอดี (ดูรูปที่ 1.4)

ในที่นี้กำหนดค่า 1 ถึง 26 สำหรับตัวอักษร A ถึง Z สมมุติว่าต้องการจะค้นว่ามีสตริง "HAVE" อยู่ในทรัยอัดแน่นหรือไม่ เริ่มต้นจากบัพรากซึ่งมีฐานอยู่ที่ตำแหน่ง 0 เราดูที่ตำแหน่ง 0+(H=8) ได้ H30 แสดงว่าถูกต้องและต้องไปที่ตำแหน่ง 30 จากตัว A เราดูที่ตำแหน่ง 30+1 ได้ A29 จากตัว V เราดูที่ตำแหน่ง 29+22=51 ได้ V2 และตัวสุดท้าย E เราดูที่ตำแหน่ง 2+5=7 ได้ E0 พร้อมทั้งตัวบ่งชี้ (flag) บอกว่าเส้นเชื่อมนี้ใช้ไปยังสถานะสุดท้าย (Final State) แสดงว่า "HAVE" อยู่ในทรัยอัดแน่นนี้ (จากการที่ตัวชี้เป็น 0 หรือว่างยังทำให้ทราบว่าไม่มีคำที่ยาวกว่านี้จาก "HAVE" แล้ว)

วิธีนี้ทำให้เราสามารถใช้เนื้อที่ที่เคยเป็นตัวชี้ว่างได้เกือบหมด แต่ทั้งนี้ต้องอาศัยการอัดบัพลงในทรัยอัดแน่นอย่างถูกวิธี เพื่อให้เนื้อที่ถูกใช้อย่างมีประสิทธิภาพ ซึ่งลักษณะนี้คล้ายคลึงกับเรื่องของการจัดสรรหน่วยความจำ (Memory Allocation)

การอัดบัพลงทรัยจะใช้วิธีแบบพอดีก่อน (First-fit) นั่นคือจะเก็บบัพทีละอันลงในทรัยโดยเริ่มต้นค้นหาช่องว่างตั้งแต่ตำแหน่ง 0 ของอาร์เรย์เพื่อหาช่องว่างที่สามารถใส่บัพนั้นได้โดยที่ไม่ทับกับเส้นเชื่อมที่มีอยู่แล้วที่ตรงนั้น และไม่ใช้ฐานเดียวกับบัพที่มีอยู่แล้ว (อาศัย BASE_USED flag บอกว่าช่องนั้นมีบัพใช้เป็นฐานแล้ว)

ลำดับในการอัดแต่ละบัพก็มีความสำคัญ ลำดับที่ดีที่สุดคือใส่บัพที่มีขนาดใหญ่ (มีเส้นเชื่อมมาก) เข้าไปก่อน แล้วค่อยใส่ขนาดตามๆกันลงมา ด้วยวิธีนี้จะทำให้มีช่องว่างเหลืออยู่น้อย เนื่องจากบัพที่มีเส้นเชื่อมน้อยจะเข้าไปอยู่ในช่องว่างที่เกิดจากบัพที่มีเส้นเชื่อมมาก และเนื่องจากบัพที่มีเส้นเชื่อมน้อย (มีเส้นเชื่อมเดียว) มีจำนวนมากพอที่จะเติมช่องว่างได้เกือบหมด

2.4 การอัดอาคม (Suffix Compression)

ข้อดีของทรัยคือการยุบอุปสรรคร่วมให้กลายเป็นทาง (path) เพียงทางเดียว ซึ่งนำไปสู่การลดขนาดได้อย่างมาก เพื่อที่จะลดขนาดลงต่อไปอีก เราจะยุบอาคม (suffix) ร่วมเสียด้วยเรียกว่า การอัดอาคม (Suffix Compression)

วิธีหนึ่งที่ทำได้คือสร้างทรัยตามปกติเสียก่อนซึ่งน่าจะเป็นทรัยโยง แล้วจึงทำการอัดอาคมในทรัยโยงนั้น (แล้วจึงใส่ลงทรัยอัดแน่นถ้าต้องการ) แต่ถ้าต้องการลดความต้องการหน่วยความจำ อาจจะใช้อัลกอริธึมอีกลักษณะในการเพิ่มคำลงทรัยโยงพร้อมกับทำการอัดอาคมเลยก็ได้

แต่ทว่าคำในภาษาไทยมีอาคม (suffix) ร่วมอยู่ค่อนข้างน้อย แม้จะเป็นในกรณีของคำประสมก็ตาม เช่น -ศาสตร์ -นิยม -วิทยา ฯลฯ ดังนั้นการทำการอัดอาคมกับคำไทยจึงได้ผลอัตราส่วนการลดขนาด (Compression Ratio) ค่อนข้างน้อย

2.5 ทรัยแถวคู่ (Double Array Trie)

เป็นการประยุกต์ใช้ทรัยอัดแน่นโดยใช้อาร์เรย์หนึ่งมิติ 2 อาร์เรย์ โดยอาร์เรย์แรกแทนฐานของทรัยอัดแน่น และอาร์เรย์ที่สองแทนเส้นเชื่อมของทรัยอัดแน่นนั้น ส่วนอาคมที่ไม่ได้ถูกยุบเป็นทางเดียว ใช้โครงสร้างเชิงรายการ (List) มาแทน ทำให้ประหยัดเนื้อที่หน่วยความจำ นอกจากนี้ยังมีคุณสมบัติจลน์ (dynamic) ในการเพิ่มลบคำของทรัยอีกด้วย

วัตถุประสงค์ของการวิจัย

เพื่อสร้างและพัฒนาโครงสร้างข้อมูลของระบบจัดการฐานข้อมูลพจนานุกรมภาษาไทยที่มีลักษณะจลน์ สามารถเพิ่มลบคำได้ตลอด เพื่อใช้ในโครงการแปลภาษาด้วยคอมพิวเตอร์ โดยให้มีประสิทธิภาพในการสืบค้นเพิ่มลบคำศัพท์ทั้งในด้านความเร็ว ความยืดหยุ่น รวมถึงการประหยัดหน่วยความจำด้วยทรัยแถวคู่

ขอบเขตของการวิจัย

ในโครงงานวิจัยนี้ จะทำการศึกษาและวิเคราะห์โครงสร้างข้อมูลทรัยแถวคู่ถึงข้อดีข้อเสียต่างๆ เพื่อนำไปใช้เป็นโครงสร้างข้อมูลของระบบจัดการฐานข้อมูลคลังศัพท์ของโครงการแปลภาษาของห้องปฏิบัติการวิจัยภาษาและวิทยาการความรู้ ศูนย์เทคโนโลยีอิเลกทรอนิกส์และคอมพิวเตอร์แห่งชาติ และเพื่อให้ระบบจัดการฐานข้อมูลคลังศัพท์ดังกล่าวนี้สามารถนำไปใช้ได้อย่างแพร่หลาย จึงเลือกที่จะพัฒนาบนระบบปฏิบัติการดอส

ในส่วนของอัลกอริธึมต่างๆที่จำเป็นต้องมี เช่น การสืบค้น การเพิ่ม และการลบคำ เป็นต้น เพื่อให้โปรแกรมประยุกต์อื่นๆสามารถเรียกใช้ได้ จะพัฒนาเป็นฟังก์ชั่นด้วยภาษาซี

ในส่วนของการทดสอบ จะทดสอบกับคำศัพท์ประเภทต่างๆ เช่น กลุ่มคำหลักในภาษาซี คำศัพท์ไทยที่ใช้บ่อย คำศัพท์อังกฤษ ในด้านความถูกต้อง และความเร็วในการสืบค้นเพิ่มลบคำในโครงสร้างข้อมูล พร้อมทั้งตรวจสอบหาขนาดของหน่วยความจำที่จำเป็นต้องใช้ โดยเปรียบเทียบกับโครงสร้างข้อมูลทรัยอัดแน่นเชิงจลน์ (Dynamic Packed Trie) ที่ทดลองสร้างขึ้น

เมื่อสิ้นสุดการทดสอบงานวิจัยในส่วนนี้ ห้องปฏิบัติการวิจัยภาษาและวิทยาการความรู้ ศูนย์เทคโนโลยีอิเลกทรอนิกส์และคอมพิวเตอร์แห่งชาติจะนำไปพัฒนาต่อโดยเพิ่มเติมฐานสำหรับจัดเก็บข้อมูลส่วนอื่นของคำ เช่น ชนิดของคำ ความหมาย ฯลฯ รวมถึงส่วนของการทำดรรชนีประเภทหลาย

ครรชนี เพื่อนำไปใช้เป็นโครงสร้างข้อมูลของระบบจัดการฐานข้อมูลคลังศัพท์กลาง และนำไปเผยแพร่ต่อไป

วิธีดำเนินการวิจัย

1. ศึกษาโครงสร้างข้อมูลทรัยแถวคู่

ศึกษาและวิเคราะห์โครงสร้างข้อมูลทรัยแถวคู่ และหาแนวทางสร้างต้นแบบของพจนานุกรมภาษาไทยโดยใช้โครงสร้างข้อมูลนี้

2. ออกแบบโครงสร้างข้อมูลทรัยแถวคู่

ศึกษาและวิเคราะห์ถึงข้อจำกัดในการสร้างโครงสร้างข้อมูลทรัยแถวคู่ เช่น ขนาดของหน่วยความจำ และทำการพัฒนาโดยใช้ภาษาซีภายใต้ระบบปฏิบัติการดอส

3. ออกแบบอัลกอริธึมการสืบค้น เพิ่ม ลบ คำศัพท์ในโครงสร้างข้อมูลทรัยแถวคู่

ออกแบบและสร้างอัลกอริธึมของการสืบค้น เพิ่ม ลบคำในโครงสร้างข้อมูล รวมถึงพัฒนาฟังก์ชั่นภาษาซีเพื่อให้สามารถเรียกใช้จากโปรแกรมประยุกต์อื่นๆได้

4. ทดสอบโครงสร้างข้อมูล และอัลกอริธึมของทรัยแถวคู่

ทำการทดสอบเพื่อตรวจสอบความถูกต้องของโครงสร้างข้อมูล และอัลกอริธึมต่างๆ โดยทดสอบกับคำ ศัพท์ประเภทต่างๆ เช่น กลุ่มคำหลักในภาษาซี คำศัพท์ไทยที่ใช้บ่อย คำศัพท์อังกฤษ ในด้านความถูกต้อง ความเร็ว ในการสืบค้น หรือเพิ่มลบคำในโครงสร้างข้อมูล และขนาดของหน่วยความจำที่ใช้ โดยเปรียบเทียบกับการใช้โครงสร้างข้อมูลทรัยอัดแน่นเชิงจลน์ที่ทดลองสร้างขึ้น

5. สรุปผลวิจัย

ประโยชน์ที่คาดว่าจะได้รับ

1. เมื่อนำไปพัฒนาต่อ โดยเพิ่มเติมข้อมูลฐานสำหรับจัดเก็บส่วนอื่นของคำ เช่น ชนิดของคำ ความหมาย ฯลฯ และส่วน ของการทำดรรชนีประเภทหลายดรรชนี จะนำไปใช้เป็นโครงสร้างข้อมูลของระบบจัด

การฐานข้อมูลคลังศัพท์ของโครงการแปลภาษาของห้องปฏิบัติการวิจัยภาษาและวิทยาการความรู้ ศูนย์เทคโนโลยีอิเลกทรอนิกส์และคอมพิวเตอร์แห่งชาติ และเผยแพร่สู่ผู้ใช้ทั่วไป

2. ใช้เป็นโครงสร้างข้อมูลของพจนานุกรมภาษาไทยมาตรฐานของโปรแกรมประยุกต์อื่นๆ เช่น พจนานุกรมสำหรับการแบ่งคำไทย การตรวจคำผิด เป็นต้น

3. สามารถนำไปพัฒนาพจนานุกรมเฉพาะทาง เช่น ทางแพทย์ ทางช่าง เป็นต้น เพื่อให้สามารถนำไปสร้างเป็นพจนานุกรมที่มีมาตรฐานยิ่งขึ้น