บทที่ 4

การทดสอบ

โครงสร้างข้อมูลทรัยอัดแน่นแบบจลน์

จากโครงสร้างข้อมูลทรัยชนิดต่างๆ พบว่าทรัยอัดแน่นเป็นโครงสร้างข้อมูลที่ดีที่สุดในแง่ของการประหยัดเนื้อที่ และอีกประการหนึ่งคือสามารถมองทรัยอัดแน่นในลักษณะเช่นเดียวกับทรัยแถวคู่ได้ คือมองเส้นเชื่อม (edge) ที่เก็บอยู่กับบัพ (node) เป็นฟังก์ชั่นการผ่าน (Transition function) ที่เรียกว่าฟังก์ชั่น goto ได้ กล่าวคือการผ่านในทรัยแถวคู่มี $g(s_r, a)$ เป็น

t = BASE[r] + a ; CHECK[t] = r

แต่เส้นเชื่อมของทรัยอัดแน่นมี $g(s_r, a)$ เป็น

t = BASE[r] + a ; CHECK[t] = a

โดย BASE[r] ต้องไม่ถูกอ้างอิงด้วยเส้นเชื่อมอื่น

จากเหตุผลสองประการดังกล่าว
จึงเลือกที่จะทดสอบเปรียบเทียบกับโครงสร้างทรัยอัดแน่นเพื่อ
หาความเหมาะสมในการประยุกต์ใช้กับพจนานุกรมภาษาไทย

ทรัยอัดแน่นที่สร้างขึ้นใหม่ได้ปรับปรุงให้เป็นทรัยอัดแน่นแบบจลน์ คือสามารถเพิ่มลบคำได้ ทำได้โดยการเพิ่ม CHECK ซึ่งใช้เก็บสถานะที่ชี้มาโดยการผ่าน a ในลักษณะเดียวกับทรัยแถวคู่ โดยแต่ละบัพในทรัยอัดแน่นแบบจลน์มีโครงสร้างข้อมูลเป็นดังนี้

```
typedef    struct      {
    LINK  next;
    LINK  back;
    CHAR q;
    CHAR dummy;
```

```
} ITEM
    typedef    long            LINK;
    typedef    unsigned char   CHAR;
```

โดยเพิ่ม dummy เข้าไปอีก 1 ไบต์สำหรับการทำเขตสำหรับคำ (Word Boundary) และ back ก็คือ CHECK ในทรัยแถวคู่นั่นเอง ซึ่งต้องมีไว้สำหรับการเพิ่มลบคีย์เข้าไปในทรัยอัดแน่นแบบจลน์นี้ และจะกำจัดทิ้งไปเมื่อแปลงทรัยอัดแน่นแบบจลน์นี้เป็นทรัยอัดแน่น(แบบสถิตย์) เพราะไม่จำเป็นต้องใช้ back ในการสืบค้นคีย์ นอกจากนี้ยังได้ปรับปรุงเพิ่มส่วน TAIL ในทรัยอัดแน่นแบบจลน์เหมือนกับของทรัยแถวคู่ เพื่อใช้เก็บสตริงเดี่ยว (Single String) เพราะการเก็บเช่นนี้ประหยัดกว่าการเก็บเป็นบัพและเส้นเชื่อม และยังสืบค้นได้เร็วกว่าเพราะเป็นเพียงการทำการเปรียบแบบ (Pattern Matching)

ในส่วนของอัลกอริธึมที่ใช้เพิ่มลบคีย์ในทรัยอัดแน่นแบบจลน์ก็เป็นทำนองเดียวกับทรัยแถวคู่ แต่มีที่ต่างกันดังนี้

1) การหาที่ว่างสำหรับบัพใหม่ใน BASE นอกจากการหาที่ว่างแบบพอดีก่อน (First-fit) สำหรับเส้นเชื่อมกที่ชี้มาจากบัพนั้นแล้ว ยังต้องตรวจสอบว่า BASE ที่อ้างด้วยเส้นเชื่อมนั้นว่างหรือไม่โดยใช้ BASE_USED flag

2) ไม่มีการเก็บ endmarker # ไว้ในทรัยอัดแน่น มีแต่ IS_LAST flag ที่ใช้บอกจุดสิ้นสุดของคำ จึงแก้ไขอัลกอริธึมที่จัดการกับ endmarker เป็นจัดการกับ IS_LAST flag แทน

3) อัลกอริธึมการสืบค้นคีย์ตรวจสอบ q[t] = a แทน CHECK[t] = r

ผลการทดสอบ

ทำการทดสอบเปรียบเทียบทรัยแถวคู่กับทรัยอัดแน่นแบบจลน์ โดยใช้คำศัพท์ประเภทต่างๆดังนี้

C คำหลักในภาษาซี

ENGLISHคำศัพท์อังกฤษ

THAI1K คำไทยใช้บ่อย 999 คำแรก

THAI10K คำไทยใช้บ่อย 10,000 คำแรก

THAI20K คำไทยใช้บ่อย 20,000 คำแรก

โดยดำเนินการทดสอบบนเครื่องไมโครคอมพิวเตอร์ที่มีหน่วยประมวลฟผลกลางอินเทล 486 ความเร็ว 33 เม็กกะเฮิตซ์ หน่วยความจำหลักขนาด 4 เม็กกะไบต์ และจำลองหน่วยความจำหลัก 3 เม็ก

กะไบต์ใช้เป็นแรมดิสก์ (RAMDisk) เก็บแฟ้มข้อมูลของคำศัพท์ต่างๆ และเก็บแฟ้มข้อมูลของโครงสร้างทรัย ผลของการทดสอบเป็นดังตารางที่ 4.1

สรุปผล

1. เวลาที่ใช้ในการเพิ่มคำ

ยิ่งคำในโครงสร้างทรัยทั้งสองยิ่งมาก เวลาในการเพิ่มคำก็ยิ่งมากขึ้นด้วย ทั้งนี้เป็นเพราะการหา BASE ที่ว่างเพื่อใส่สถานะและการผ่านแบบพอดีก่อน (First-fit) นั้นจะต้องเริ่มหาตั้งแต่ BASE แรกของทรัย และพบว่าทรัยอัดแน่นแบบจลน์ใช้เวลาในการเพิ่มคำมากกว่าทรัยแถวคู่ เพราะว่าต้องตรวจสอบว่า BASE ที่จะใช้นั้นถูกอ้างอิงโดยการผ่านอื่นหรือไม่ ซึ่งทำให้เสียเวลามากขึ้น

2. เวลาที่ใช้ในการสืบค้นคำ

เช่นเดียวกับการเพิ่มคำคือ คำในทรัยยิ่งมาก เวลาในการสืบค้นก็มากขึ้น ทั้งที่เวลาสืบค้นเฉลี่ยควรจะเท่าๆกัน ทั้งนี้เป็นเพราะทรัยอยู่ในหน่วยความจำสำรองซึ่งมีโครงสร้างการเก็บข้อมูลในระบบปฏิบัติการดอสเป็นแบบรายการโยง (Linked List) โดยใช้ตารางจัดสรรแฟ้มข้อมูล (File Allocation Table —— FAT) และพบว่าทรัยอัดแน่นแบบจลน์และทรัยแถวคู่ใช้เวลาสืบค้นเท่าๆกัน เนื่องจากมีการตรวจสอบการผ่านที่คล้ายกัน คือในทรัยอัดแน่นแบบจลน์ใช้ q[t]=a และในทรัยแถวคู่ใช้ CHECK[t]=r

3. เนื้อที่หน่วยความจำที่ใช้

อัตราส่วนระหว่างเนื้อที่หน่วยความจำของทรัยต่อเนื้อที่หน่วยความจำของแฟ้มของคำแปรผกผันกับจำนวนคำ กล่าวคือยิ่งคำมีจำนวนมาก ยิ่งมีการใช้ประสิทธิภาพของเนื้อที่หน่วยความจำของทรัยดีขึ้น และพบว่าอัตราส่วนระหว่างเนื้อที่หน่วยความจำของทรัยต่อเนื้อที่หน่วยความจำของแฟ้มของคำของทรัยอัดแน่นแบบจลน์น้อยกว่าของทรัยแถวคู่ เป็นเพราะเหตุผลเดียวกับเวลาที่ใช้ในการเพิ่มคำ คือการที่ต้องตรวจสอบ BASE ว่าว่างหรือไม่ แต่ถ้าหากแปลงทรัยอัดแน่นแบบจลน์เป็นทรัยอัดแน่น(แบบสถิตย์)ที่ไม่ต้องมี back แล้วจะทำให้อัตราส่วนดังกล่าวน้อยลง

4. ความกระชับของเนื้อที่หน่วยความจำ

ความกระชับของเนื้อที่หน่วยความจำวัดจาก (เนื้อที่ที่ใช้ - เนื้อที่ที่ว่าง) / เนื้อที่ที่ใช้ พบว่าในส่วนของ TAIL มีความกระชับที่ใกล้เคียงกันไม่เปลี่ยนไปตามจำนวนของคำ แต่ส่วนของอาร์เรย์คู่และของทรัยอัดแน่นแบบจลน์แปรตามจำนวนคำคือจำนวนคำยิ่งมาก ประสิทธิภาพการใช้หน่วยความจำยิ่งดี นอกจากนี้ยังพบว่าความกระชับของทรัยอัดแน่นแบบจลน์ดีกว่าของทรัยแถวคู่ เป็นเพราะการที่ทรัยอัด

แน่นแบบจลน์ใช้ IS_LAST เป็นการบอกการสิ้นสุดสตริงแทนการใช้ endmarker ในทรัยแถวคู่ จึงไม่สิ้นเปลือง BASE ใหม่สำหรับการผ่าน (Transition) ของ endmarker ของทุกคำ

5. จากผลการทดสอบจึงกล่าวได้ว่าทรัยแถวคู่เหมาะที่จะใช้เป็นโครงสร้างข้อมูลสำหรับฐานข้อมูลพจนานุกรมภาษาไทยที่จำเป็นต้องมีการเพิ่มลบคำ เพราะว่ามีประสิทธิภาพการใช้เนื้อที่หน่วยความจำได้ดี และมีเวลาที่ใช้ในการสืบค้นเพิ่มลบข้อมูลที่ค่อนข้างดี จึงยืดหยุ่นพอที่จะนำไปใช้เป็นโครงสร้างข้อมูลของพจนานุกรมสำหรับการแบ่งคำด้วย ถ้าหากผู้ใช้ต้องการที่จะทำการแก้ไขพจนานุกรมของตนเอง

	Double Array Trie					Dynamic Packed Trie				
	C	ENGLISH	THAI1K	THAI10K	THAI20K	C	ENGLISH	THAI1K	THAI10K	THAI20K
จำนวนคำ	32	19,816	999	10,000	20,000	32	19,816	999	10,000	20,000
ขนาดไฟล์คำศัพท์ (ไบต์)	230	195,914	5,951	77,274	168,904	230	195,914	5,951	77,274	168,904
ขนาดไฟล์ทรัย (ไบต์)	2308	349,247	16,870	158,835	324,189	2,260	376,830	18,468	169,567	347,943
ทรัย / คำศัพท์	10.03	1.78	2.83	2.06	1.92	9.82	1.92	3.10	2.19	2.06
จำนวนโหนด	271	35,893	1,784	15,896	31,837	212	31,476	2,598	13,790	27,845
จำนวนโหนดที่ใช้	50	35,705	1,535	15,431	31,245	163	31,425	2,055	13,454	27,552
ความกระชับของโหนด (%)	18.45	99.48	86.04	97.07	98.14	76.89	99.84	79.10	97.56	98.95
ขนาด TAIL (ไบต์)	140	62,103	2,598	31,667	69,493	140	62,070	1,587	31,667	69,493
ขนาด TAIL ที่ใช้ (ไบต์)	123	47,196	2,055	26,456	58,720	123	47,157	1,332	26,456	58,720
ความกระชับของ TAIL (%)	87.86	76.00	79.10	83.54	84.50	87.86	75.97	83.93	83.54	84.50
- สืบค้นเฉลี่ย (มิลลิวินาที)	1	9	3	6	11	1	9	3	6	11
- สืบค้นมากสุด (มิลลิวินาที)	6	54	11	22	40	6	54	11	22	40
- เพิ่มคำเฉลี่ย (มิลลิวินาที)	10	540	30	220	410	10	600	40	260	490
- เพิ่มคำมากสุด (มิลลิวินาที)	60	7,470	110	1,650	5,270	60	7,750	490	1,870	7,410

ตารางที่ 4.1 ผลการทดสอบกับคำประเภทต่างๆ