บทที่ 3

การออกแบบโครงสร้างข้อมูลของดัชนี

ความหมายของคำว่าดัชนีที่ใช้ในงานวิจัยนี้หมายถึงแอตทริบิวท์ที่ใช้ในการค้นหาข้อมูล ซึ่งในงานวิจัยนี้ใช้แอตทริบิวท์ที่ไม่ใช่คีย์หลักเป็นดัชนีในการค้นหา ส่วนความหมายของคำว่าข้อมูลของดัชนีหมายถึงแอตทริบิวท์อื่น ๆ นอกเหนือจากแอตทริบิวท์ที่ใช้เป็นดัชนี

3.1 ลักษณะโครงสร้างข้อมูลของดัชนี

โครงสร้างข้อมูลของดัชนีที่นำมาใช้ในงานวิจัยนี้ได้แก่โครงสร้างข้อมูลแบบทู-ทรัย[1] เนื่องจากโครงสร้างข้อมูลแบบทู-ทรัยมีลักษณะเด่นต่าง ๆ ที่เหมาะกับงานวิจัยดังนี้

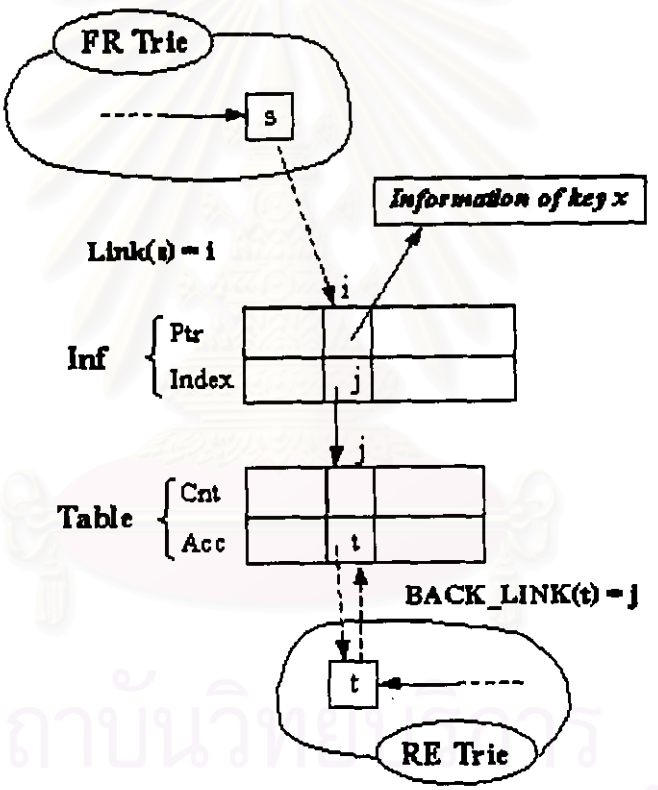
1. สามารถระบุข้อมูลของแต่ละดัชนีที่จัดเก็บได้อย่างเป็นเอกลักษณ์
2. สามารถใช้ได้กับกลุ่มของข้อมูลที่มีลักษณะแบบพลวัตซึ่งหมายถึงกลุ่มข้อมูลที่สามารถเพิ่มข้อมูลใหม่ หรือลบข้อมูลที่มีอยู่ในกลุ่มข้อมูลได้ โดยมีขั้นตอนวิธีในการปรับโครงสร้างข้อมูลที่มีประสิทธิภาพในการขจัดหน่วยความจำที่ไม่ได้ใช้งาน
3. การค้นหาข้อมูลในโครงสร้างข้อมูลแบบทู-ทรัยสามารถค้นหาได้ทีละตัวอักษรจึงสามารถทำการค้นหาแบบส่วนเพิ่ม (Incremental Search) ได้โดยผู้ใช้สามารถพิมพ์ดัชนีที่ใช้ในการค้นหาทีละตัวอักษร เมื่อส่วนของดัชนีที่พิมพ์นั้นแตกต่างจากดัชนีอื่นที่เก็บในโครงสร้างข้อมูลแล้ว โครงสร้างข้อมูลจะสามารถระบุข้อมูลของส่วนของดัชนีนั้นได้ทันที โดยที่ผู้ใช้ไม่ต้องพิมพ์ดัชนีที่ใช้ค้นหาทั้งหมด
4. โครงสร้างข้อมูลแบบทู-ทรัยจัดเก็บส่วนเติมหน้า (Prefix) และส่วนเติมหลัง (Suffix) ของแต่ละดัชนีที่เหมือนกันไว้เพียงที่เดียว จึงทำให้ค้นหาแบบส่วนเติมหน้าและค้นหาแบบส่วนเติมหลังได้ อีกทั้งช่วยทำให้ประหยัดเนื้อที่ของโครงสร้างข้อมูลด้วย

---

[1] Jun-ichi Aoe, Katsushi Morimoto, Masami Shishibori, and Ki-Hong Park, A Trie Compaction Algorithm for a Large Set of Keys, IEEE Transactions on Knowledge and Data Engineering, Vol. 8, June 1996, pp. 476-491.

โครงสร้างข้อมูลแบบทู-ทรัยในเชิงโปรแกรมแบ่งเป็น 2 ลักษณะได้แก่โครงสร้างเชิงรายการและโครงสร้างแถวลำดับคู่ ซึ่งการเพิ่มและลบข้อมูลในโครงสร้างเชิงรายการสามารถทำได้รวดเร็วกว่าโครงสร้างแถวลำดับคู่แต่โครงสร้างแถวลำดับคู่จะมีประสิทธิภาพในการค้นหาข้อมูลดีกว่าโครงสร้างเชิงรายการ

ในงานวิจัยนี้เลือกใช้โครงสร้างแถวลำดับคู่ในการสร้างโครงสร้างข้อมูลแบบทู-ทรัยในเชิงโปรแกรม เนื่องจากส่วนการทำงานหลักของงานวิจัยนี้จะเน้นการค้นหาข้อมูลโดยใช้แอตทริบิวท์ที่ไม่ใช่คีย์หลักมากกว่าการทำการเพิ่มและลบข้อมูลในโครงสร้างข้อมูล ลักษณะโครงสร้างข้อมูลแบบทู-ทรัยโดยใช้โครงสร้างแถวลำดับคู่ในเชิงโปรแกรม[2] แสดงดังรูปที่ 3.1



รูปที่ 3.1 โครงสร้างแบบทู-ทรัยโดยใช้โครงสร้างแถวลำดับคู่ในเชิงโปรแกรม

โครงสร้างข้อมูลแบบทู-ทรัยในเชิงโปรแกรมประกอบด้วยโครงสร้างแถวลำดับคู่จำนวน 4 แถวได้แก่ (1)แถวลำดับคู่ของทรัยส่วนหน้า (2)แถวลำดับคู่ของทรัยส่วนหลัง (3)แถวลำดับคู่ของข้อมูล (Inf) และ (4)แถวลำดับคู่เทเบิล (Table) โดยที่แถวลำดับคู่ของทรัยส่วนหน้าและแถว

[2] Ibid., p. 484.

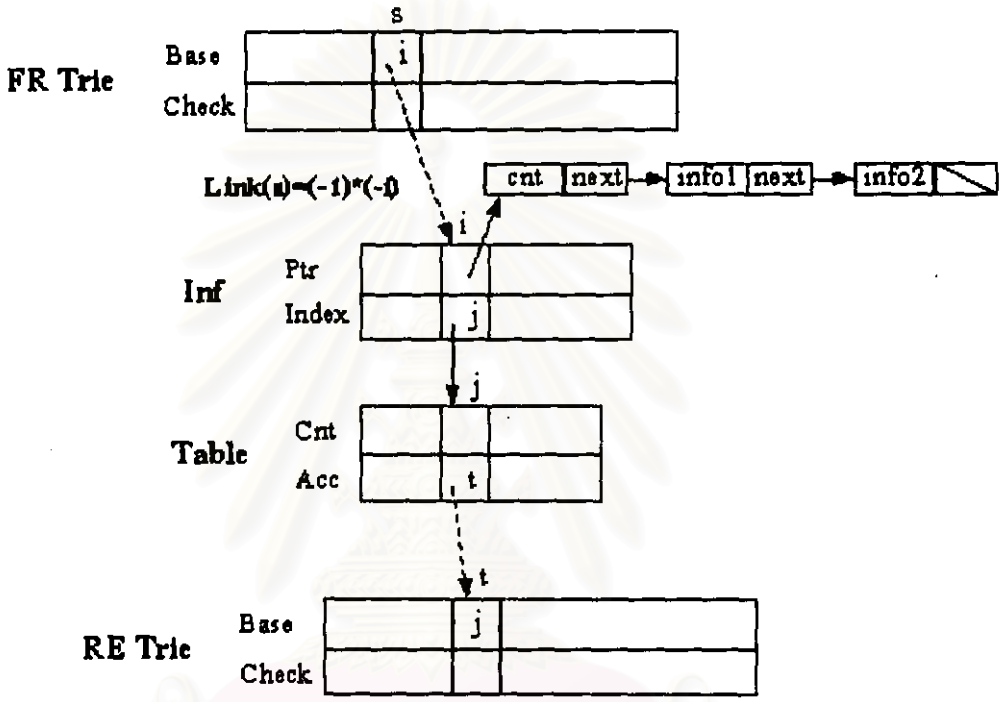
ลำดับคู่ของทรัยส่วนหลังประกอบด้วยอะเรย์ (Array) ขนาด 1 มิติ จำนวน 2 อะเรย์คืออะเรย์เบส (Base) และอะเรย์เช็ค (Check) และมีหมายเลขประจำโหนดเป็นดัชนีเพื่อแสดงความสัมพันธ์ระหว่างเบสและเช็ค แถวลำดับคู่ของข้อมูลประกอบด้วยอะเรย์พอยท์เตอร์ (Ptr) และอะเรย์อินเดกซ์ (Index) และแถวลำดับคู่เทเบิลประกอบด้วยอะเรย์เคาท์เตอร์ (Cnt) และอะเรย์แอคเซพติง (Acc)

รายละเอียดของอะเรย์แต่ละชนิด มีดังนี้

1. อะเรย์เบส ประกอบด้วย 2 ส่วนคือ หมายเลขตำแหน่งของอะเรย์และค่าตัวเลขที่เก็บในอะเรย์เบส หมายเลขตำแหน่งของอะเรย์ใช้แทนหมายเลขโหนด ส่วนค่าตัวเลขที่เก็บในอะเรย์แบ่งเป็น 3 ประเภทคือ (1)ถ้ามีค่าบวกจะเป็นตัวเลขที่ใช้สำหรับคำนวณเส้นทางเดินจากโหนดหนึ่งไปยังอีกโหนดหนึ่ง (2)ถ้าค่าตัวเลขเป็นศูนย์แสดงว่าอะเรย์นั้นว่างอยู่ (3)แต่ถ้ามีค่าเป็นลบในแถวลำดับคู่ของทรัยส่วนหน้าจะเป็นตำแหน่งของเซพพาเรทโหนดและเป็นตำแหน่งของแถวลำดับคู่ข้อมูลด้วย ส่วนค่าตัวเลขลบของอะเรย์เบสในแถวลำดับคู่ของทรัยส่วนหลังจะเป็นตำแหน่งของแอคเซพติงโหนดและเป็นตำแหน่งของแถวลำดับคู่เทเบิลด้วย
2. อะเรย์เช็ค ประกอบด้วย 2 ส่วนคือ หมายเลขตำแหน่งของอะเรย์และค่าตัวเลขที่เก็บในอะเรย์เช็ค หมายเลขตำแหน่งของอะเรย์ใช้แทนหมายเลขโหนด ส่วนค่าตัวเลขที่เก็บในอะเรย์คือตำแหน่งของโหนดแม่ (Parent Node) เช่นโหนดหมายเลข 3 สร้างเส้นทางเดินไปยังโหนดที่ 5 นั่นคือโหนดหมายเลข 3 เป็นโหนดแม่ของโหนดหมายเลข 5 ดังนั้นค่าของอะเรย์เช็คที่ 5 มีค่าเป็น 3
3. อะเรย์พอยท์เตอร์ ใช้เก็บพอยท์เตอร์ที่ชี้ไปยังข้อมูลของแต่ละดัชนี
4. อะเรย์อินเดกซ์ ใช้เก็บตำแหน่งของแถวลำดับคู่เทเบิล เพื่อใช้ในการลิงค์จากแถวลำดับคู่ของทรัยส่วนหน้าไปยังแถวลำดับคู่ของทรัยส่วนหลัง
5. อะเรย์เคาท์เตอร์ ใช้เก็บจำนวนลิงค์ของแต่ละเซพพาเรทโหนดของทรัยส่วนหน้าที่ลิงค์มายังแอคเซพติงโหนดของทรัยส่วนหลัง
6. อะเรย์แอคเซพติง ใช้เก็บตำแหน่งของแอคเซพติงโหนด และเป็นตำแหน่งของแอคเซพติงโหนดในแถวลำดับคู่ของทรัยส่วนหลังด้วย

จากงานวิจัยโครงสร้างข้อมูลแบบทู-ทรัยของ Jun-ichi Aoe, Katsushi Morimoto, Masami Shishibori และ Ki-Hong Park ไม่ได้ระบุลักษณะการจัดเก็บข้อมูลของแต่ละดัชนีไว้ ผู้วิจัยจึงได้ออกแบบส่วนของการจัดเก็บข้อมูลของแต่ละดัชนีให้เหมาะสมกับงานวิจัยที่พัฒนาขึ้น

เนื่องจากงานวิจัยนี้ใช้แอตทริบิวท์ที่ไม่ใช่คีย์หลักเป็นดัชนีในการค้นหาข้อมูลทำให้ข้อมูลของแต่ละดัชนีที่จัดเก็บสามารถมีจำนวนข้อมูลมากกว่า 1 ข้อมูลได้ ดังนั้นจึงได้ทำการออกแบบส่วนของการจัดเก็บข้อมูลของแต่ละดัชนีเป็นลักษณะของลิงค์ลิสต์ (Link List) โดยโหนดแรกของลิงค์ลิสต์ใช้เก็บจำนวนข้อมูลทั้งหมดของดัชนีนั้น ส่วนโหนดที่เหลือทั้งหมดใช้เก็บข้อมูลของดัชนี ซึ่งลักษณะของโครงสร้างข้อมูลแถวลำดับคู่ที่ออกแบบไว้จะแสดงได้ดังรูปที่ 3.2



รูปที่ 3.2 โครงสร้างแถวลำดับคู่ที่ได้ออกแบบส่วนการจัดเก็บข้อมูลของแต่ละดัชนี

3.2 ขั้นตอนวิธีของโครงสร้างข้อมูลของดัชนี

ขั้นตอนวิธีของโครงสร้างข้อมูลของดัชนีที่ใช้ในงานวิจัยได้แก่ ขั้นตอนวิธีค้นหาข้อมูลโดยใช้ดัชนี ขั้นตอนวิธีเพิ่มและลบดัชนีและข้อมูลของดัชนีในงานวิจัยของ Jun-ichi Aoe, Katsushi Morimoto, Masami Shishibori และ Ki-Hong Park แต่มีฟังก์ชันบางฟังก์ชันที่ผู้วิจัยได้ทำการปรับและสร้างเพิ่มจากงานวิจัยเดิมเพื่อทำให้โครงสร้างข้อมูลในเชิงโปรแกรมทำงานได้ดีขึ้น โดยแต่ละขั้นตอนวิธีจะต้องใช้สมการของโครงสร้างแถวลำดับคู่ในการอ้างอิงตำแหน่งของแต่ละโหนดที่เก็บในโครงสร้างแถวลำดับคู่

สมการที่ใช้ในการจัดการกับข้อมูลที่เก็บอยู่ในโครงสร้างแถวลำดับคู่ ได้แก่

$$t = Base[s] + a_h$$
$$Check[t] = s$$

กำหนดให้ s คือหมายเลขโหนดปัจจุบัน

t คือหมายเลขโหนดถัดไป

a_h คือค่าตัวเลขที่ใช้แทนตัวอักษรลำดับที่ h (ในงานวิจัยนี้ใช้รหัสแอสกีของตัวอักษร)

ในการตรวจสอบว่ามีตัวอักษร a (ทรานสิชั่น a) ระหว่างโหนด s และโหนด t หรือไม่ ให้เปรียบเทียบว่า Check [Base[s] + a] = s หรือไม่ หากเท่ากันแสดงว่ามีตัวอักษร a อยู่ระหว่างโหนด s และโหนด t จริง ดังนั้นในการค้นหาตัวอักษรหนึ่ง ๆ จะใช้เวลามากที่สุด (Worst-Case Time Complexity)[3] เป็น O(1)

จากสมการที่ใช้ในการจัดการกับข้อมูลที่เก็บอยู่ในโครงสร้างแถวลำดับคู่สามารถนำมาประยุกต์ใช้กับโครงสร้างข้อมูลแบบทู-ทรัยได้ดังนี้

แถวลำดับคู่ของทรัยส่วนหน้า: $t = FrBase[s] + a_h$

$FrCheck[t] = s$

แถวลำดับคู่ของทรัยส่วนหลัง: $t = ReBase[s] + a_h$

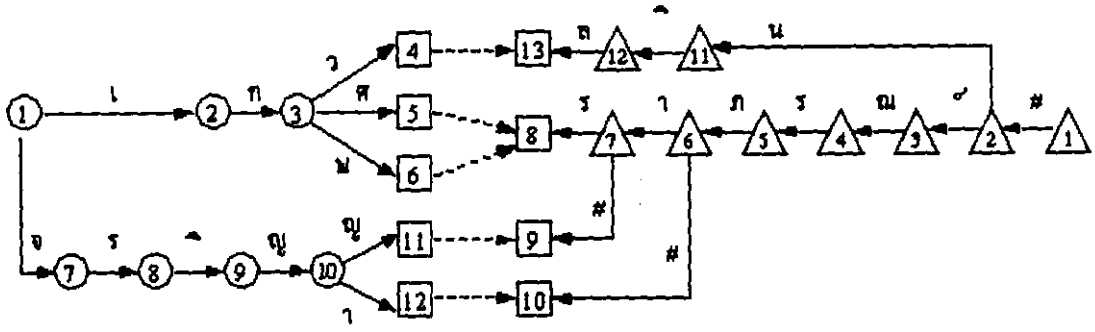
$ReCheck[t] = s$

ตัวอย่างดัชนีที่นำมาสร้างเป็นโครงสร้างข้อมูลแบบทู-ทรัย ได้แก่ "เกวลิน" "เกศรากรณ์" "เกษรากรณ์ " "จริญญากรณ์ " และ "จริญากรณ์" โดยใช้ "#" เป็นตัวสิ้นสุดของแต่ละดัชนี ลักษณะโครงสร้างข้อมูลแบบทู-ทรัยของตัวอย่างแสดงดังรูปที่ 3.3

ก่อนที่จะทำการค้นหาดัชนี เพิ่มดัชนีและข้อมูลของดัชนี รวมทั้งลบดัชนีและข้อมูลของดัชนีที่อยู่ในโครงสร้างข้อมูลแบบทู-ทรัย จะต้องเพิ่ม "#" เป็นตัวปิดท้ายดัชนีนั้น

---

[3] วิรัช ศรเลิศล้ำวาณิช, อภิชิต พิทยรัตน์โสภณ และเกรียงชัย จันทร์แสนวิไล, การจัดการฐานข้อมูลทงนานุกรมไทยด้วยทรัยแถวคู่หลายดรรชนี, การประชุมวิชาการ ครั้งที่ 5, ศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ, หน้า 197-206.



รูปที่ 3.3 โครงสร้างข้อมูลแบบทู-ทรัยของตัวอย่างดัชนี

3.2.1 ขั้นตอนวิธีค้นหาข้อมูลโดยใช้ดัชนี

การค้นหาข้อมูลในโครงสร้างแถวลำดับคู่เป็นวิธีการค้นหาโดยใช้อักษรแต่ละตัวที่ประกอบเป็นดัชนีนั้น ๆ เข้าไปค้นหาในตาราง (อะเรย์ขนาด 1 มิติจำนวน 2 อะเรย์) ซึ่งจะได้ผลลัพธ์เป็นค่าฐานเริ่มต้นสำหรับตัวอักษรตัวต่อไป และในการค้นหาต่อจะดำเนินการค้นหาเรื่อยไปจนถึงตัวสุดท้าย จะได้ผลลัพธ์เป็นค่าบอกตำแหน่งของข้อมูลสำหรับดัชนีนั้น ๆ ความเร็วในการค้นหาจึงขึ้นกับจำนวนตัวอักษรของดัชนีนั้น ๆ โดยไม่ขึ้นกับจำนวนคำที่ประกอบขึ้นเป็นตารางทรัยแถวลำดับคู่ ดังนั้นการค้นหาดัชนีซึ่งประกอบด้วย k ตัวอักษรจะใช้เวลาในการค้นหามากที่สุด[4] เท่ากับ O(k)

ขั้นตอนวิธีการค้นหาข้อมูลโดยใช้ดัชนี มีดังนี้

1. กำหนดค่าเริ่มต้น $s = 1$ และ $h = 0$
2. นำอักษรตัวแรก a_1 มาทำการค้นหา โดยมีค่า $s = 1$ โดยใช้สมการ

   $t = \text{FrBase}[s] + a_h$

   ในการค้นหาอักษรตัวถัดไป ต้องให้ค่าเริ่มต้น s เท่ากับค่า t ของตัวอักษรปัจจุบัน
3. ตรวจสอบว่าโหนดที่สร้างขึ้น (t) มีค่าเกินขนาดของแถวลำดับคู่ของทรัยส่วนหน้าหรือไม่ ถ้ามากกว่าขนาดของอะเรย์ให้ระบุว่าไม่พบดัชนีและออกจากการค้นหา
4. ตรวจสอบว่าโหนดที่สร้างใหม่เป็นโหนดที่สร้างจากโหนดปัจจุบัน (s) หรือไม่

---

[4] Jun-ichi Aoe, Katsushi Morimoto, Masami Shishibori, and Ki-Hong Park, A Trie Compaction Algorithm for a Large Set of Keys, IEEE Transactions on Knowledge and Data Engineering, Vol. 8, June 1996, pp. 476-491.

- ถ้า FrCheck[t] = s แสดงว่าโหนดใหม่สร้างจากโหนดปัจจุบัน

- ถ้า FrCheck[t] != s แสดงว่าโหนดใหม่ไม่ได้สร้างจากโหนดปัจจุบัน ระบุว่าไม่พบดัชนีและออกจากการค้นหา

5. ตรวจสอบว่าโหนดที่สร้างใหม่เป็นโหนดประเภทใดโดยพิจารณาจากค่า FrBase [s]

   - ถ้า FrBase[s] = 0 แสดงว่าเป็นโหนดที่ยังไม่ได้ใช้งาน ระบุว่าไม่พบดัชนีและออกจากการค้นหา

   - ถ้า FrBase[s] > 0 แสดงว่าเป็นโหนดที่อยู่ในเส้นทางระหว่างโหนดแรกและเซพพาเรทโหนด

   - ถ้า FrBase[s] < 0 แสดงว่าเป็นเซพพาเรทโหนด จะใช้ฟังก์ชัน ReadReTrie(t) ในการทำการหาส่วนที่เหลือของดัชนีที่อยู่ในแถวลำดับคู่ของทรัยส่วนหลังทั้งหมด จากนั้นนำมาเปรียบเทียบกับส่วนที่เหลือของดัชนี ถ้าไม่เท่ากันระบุว่าไม่พบดัชนีและออกจากการค้นหา ถ้าเท่ากัน แสดงว่าพบดัชนีและใช้ฟังก์ชัน GetInfPtr(i) ค้นหาข้อมูลของดัชนีนั้น โดย i คือหมายเลขตำแหน่งของอะเรย์ที่เก็บข้อมูล จากนั้นจะส่งข้อมูลกลับมา

6. จากข้อ 5 กรณีค่า FrBase[s] > 0 สามารถค้นหาโหนดต่อไปได้ โดยใช้โหนดที่สร้างใหม่แทนโหนดปัจจุบัน แล้วทำงานซ้ำตั้งแต่ข้อ 2 ถึง 5 จนกระทั่งตัวอักษรที่ประกอบเป็นดัชนีหมด หรือออกจากการค้นหา

- ฟังก์ชัน ReadReTrie(t) โดย t คือหมายเลขดัชนีของแอคเซพติงโหนด

ใช้ในการหาส่วนที่เหลือของดัชนีที่เก็บในแถวลำดับคู่ของทรัยส่วนหลังทั้งหมด โดยเริ่มหาจากแอคเซพติงโหนดไปเรื่อย ๆ จนถึงโหนด # ซึ่งเป็นตัวปิดท้ายดัชนีนั้น

- ฟังก์ชัน GetInfPtr(i)

เป็นฟังก์ชันที่ผู้วิจัยสร้างเพิ่มจากงานวิจัยเดิม เพื่อใช้ในการหาข้อมูลทั้งหมดของดัชนีที่มีลักษณะเป็นลิงค์ลิสต์ในอะเรย์ข้อมูล โดย i คือหมายเลขตำแหน่งของอะเรย์ข้อมูลที่ได้จากค่า FrBase[spn] คูณด้วย -1

3.2.2 ขั้นตอนวิธีเพิ่มดัชนีและข้อมูลของดัชนี

ก่อนที่จะทำการเพิ่มดัชนีหรือข้อมูลของดัชนีในโครงสร้างข้อมูลแบบทู-ทรัยจะต้องทำการค้นหาดัชนีและข้อมูลของดัชนีนั้นก่อน เมื่อไม่พบจึงทำการเพิ่มดัชนีและข้อมูลของดัชนี การ

เพิ่มดัชนีในโครงสร้างข้อมูลแบบทู-ทรัยของเครื่องมือที่พัฒนาขึ้นแบ่งเป็น 2 แบบคือ การเพิ่มดัชนีและข้อมูลของดัชนีใหม่ และการเพิ่มข้อมูลของดัชนีที่มีอยู่เดิมเนื่องจากดัชนีที่ใช้เป็นแอตทริบิวท์ที่ไม่ใช่คีย์หลักจึงทำให้ข้อมูลของดัชนีหนึ่งสามารถมีจำนวนมากกว่า 1 ข้อมูล ดังนั้นเมื่อทำการค้นหาแล้วพบดัชนีแต่ไม่พบข้อมูลของดัชนี จึงทำการเพิ่มเฉพาะข้อมูลใหม่ของดัชนีที่มีอยู่เดิม

การค้นหาข้อมูลแล้วไม่พบดัชนีแบ่งเป็น 2 กรณี ดังนี้

(1) เมื่อค้นหาดัชนีในแถวลำดับคู่ของทรัยส่วนหน้าแล้วพบว่าโหนดที่ใช้ค้นหาไม่ได้สร้างจากโหนดปัจจุบันที่มีอยู่

(2) เมื่อทำการค้นหาไปจนถึงเซพพาเรทโหนดและนำส่วนที่เหลือในแถวลำดับคู่ของทรัยส่วนหลังมาเปรียบเทียบกับส่วนที่เหลือของดัชนีแล้ว พบว่าไม่เท่ากัน

ดังนั้นการเพิ่มดัชนีและข้อมูลของดัชนีจึงได้แบ่งออกเป็น 2 กรณีคือการเพิ่มส่วนของดัชนีที่ไม่มีอยู่ในแถวลำดับคู่ของทรัยส่วนหน้า และการเพิ่มส่วนของดัชนีที่ไม่ตรงกับส่วนที่มีอยู่ในแถวลำดับคู่ของทรัยส่วนหลัง รายละเอียดของแต่ละกรณีมีดังนี้

3.2.2.1 การเพิ่มส่วนของดัชนีที่ไม่พบในแถวลำดับคู่ของทรัยส่วนหน้า

เนื่องจากค้นหาในแถวลำดับคู่ของทรัยส่วนหน้าแล้วพบว่าโหนดที่ค้นหาไม่ได้สร้างจากโหนดปัจจุบัน ซึ่งการเพิ่มส่วนของดัชนีที่ไม่พบในแถวลำดับคู่ของทรัยส่วนหน้าใช้ฟังก์ชันต่าง ๆ ดังนี้

กำหนดให้ s แทนหมายเลขโหนดปัจจุบัน

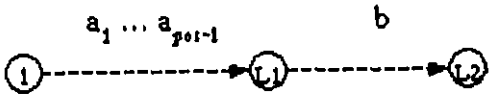
$a_{pos}a_{pos+1}\ldots\ldots a_n a_{n+1}$ แทนส่วนที่เหลือของดัชนีที่จะต้องทำการเพิ่ม

- ฟังก์ชัน FrInsert(s, $a_{pos}a_{pos+1}\ldots\ldots a_n a_{n+1}$) จะทำการเพิ่ม a_{pos} ในแถวลำดับคู่ของทรัยส่วนหน้าและเพิ่มส่วนที่เหลือ $a_{pos+1}\ldots\ldots a_n a_{n+1}$ ในแถวลำดับคู่ของทรัยส่วนหลังซึ่งมีขั้นตอนการทำงานดังนี้

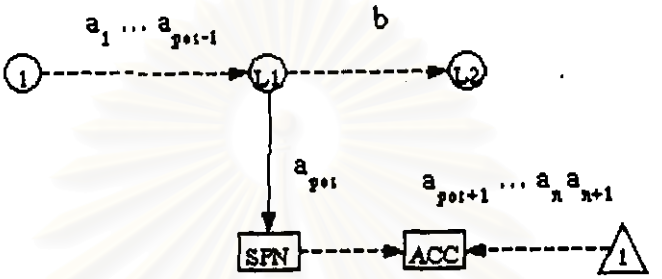
  1. กำหนดค่าของหมายเลขโหนดถัดไปโดยใช้สมการ $t = FrBase[s] + a_{pos}$
  2. ตรวจสอบค่าของ FrCheck[t] ถ้ามีค่าไม่เท่ากับ 0 ทำงานในข้อ 3
     ถ้าค่าของ FrCheck[t] เท่ากับ 0 ทำงานในข้อ 4

3. กรณีที่ค่าของ FrCheck[t] ไม่เท่ากับ 0 แสดงว่าเกิดการชน (Collision) กับโหนดที่มีอยู่เดิม จึงต้องทำการย้ายโหนดโดยเปลี่ยนค่าของ FrBase[s] และ FrBase[t] ใหม่ ซึ่งมีขั้นตอนดังนี้
   3.1 โดยใช้ฟังก์ชัน SetFrList(s) เพื่อหาลิสต์ (List) ของโหนดต่าง ๆ ที่โหนด s ลิงค์ไปหา ซึ่งจะเก็บไว้ใน Slist และใช้ฟังก์ชัน SetFrList (FrCheck[t]) เพื่อหา Tlist
   3.2 ตรวจสอบจำนวนสมาชิกของ Slist + 1 ว่ามีน้อยกว่าจำนวนสมาชิกของ Tlist หรือไม่ ถ้าเงื่อนไขเป็นจริง ทำงานในข้อ 3.3 แต่ถ้าเงื่อนไขเป็นเท็จ ทำงานในข้อ 3.4
   3.3 ใช้ฟังก์ชัน FrChange(s, Slist, a_{pos}) เพื่อย้ายโหนดต่าง ๆ ใน Slist ไปยังที่อยู่ใหม่ที่ไม่ทำให้เกิดการชนกับโหนดใหม่ที่เพิ่มมา โดยจะต้องหาค่า FrBase[s] ใหม่ ซึ่งจะเรียกใช้ฟังก์ชัน FrXCheck(Slist) ในการหาค่า FrBase[s] ที่ทำให้โหนด s สามารถลิงค์ไปหาสมาชิกทุกตัวใน Slist รวมทั้ง a_{pos} ด้วย และต้องอยู่ภายใต้เงื่อนไข FrCheck [FrBase[s] + b] = 0 โดยที่ b เป็นสมาชิกใน Slist U $\{a_{pos}\}$ จากนั้นทำงานในข้อ 4
   3.4 ใช้ฟังก์ชัน FrChange(t, Tlist) เพื่อย้ายโหนดต่าง ๆ ใน Tlist ไปยังที่อยู่ใหม่ที่ไม่ทำให้เกิดการชนกับโหนดใหม่ที่เพิ่มมา โดยจะต้องหาค่า FrBase[t] ใหม่ ซึ่งจะเรียกใช้ฟังก์ชัน FrXCheck(Tlist) ในการหาค่า FrBase[t] ที่ทำให้โหนด t สามารถลิงค์ไปหาสมาชิกทุกตัวใน Tlist และต้องอยู่ภายใต้เงื่อนไข FrCheck [FrBase[t] + b] = 0 โดยที่ b เป็นสมาชิกใน Tlist จากนั้นทำงานในข้อ 4
4. ใช้ฟังก์ชัน WriteInfPtr(n) ทำการเก็บข้อมูลของดัชนีไว้ในอะเรย์ของข้อมูล โดย n คือหมายเลขตำแหน่งของอะเรย์ของข้อมูลที่ว่างอยู่ ซึ่งได้มาจากการเรียกใช้ฟังก์ชัน GetAvailInfIndx()
5. ใช้ฟังก์ชัน GetAvailTableIndx() เพื่อหาหมายเลขตำแหน่งของอะเรย์เทเบิลที่ว่างซึ่งเท่ากับค่า m และกำหนดค่า InfIndex[n] = m
6. กำหนดค่า TableCnt[m] = TableCnt[m] + 1
7. ใช้ฟังก์ชัน ReAdd(a_{pos+1}..... $a_n a_{n+1}$) เพื่อสร้างส่วนที่เหลือของดัชนีในแถวลำดับคู่ของทรัยส่วนหลัง และคืนค่าหมายเลขดัชนีของแอดเซพติงโหนดกลับมา
8. กำหนดค่า TableAcc[m] เท่ากับหมายเลขดัชนีของแอดเซพติงโหนดที่ได้จากฟังก์ชัน ReAdd(a_{pos+1}..... $a_n a_{n+1}$)

การทำงานของฟังก์ชัน FrInsert(s, $a_{pos}a_{pos+1}$..... $a_n a_{n+1}$) แสดงดังรูปที่ 3.4



(ก) ทรัยส่วนหน้า ก่อนทำการเพิ่มดัชนี



(ข) ทรัยส่วนหน้า หลังทำการเพิ่มดัชนี

รูปที่ 3.4 การทำงานของฟังก์ชัน FrInsert

- ฟังก์ชัน SetFrList(s)

  ใช้ในการหาลิสต์ที่เก็บโหนดต่าง ๆ ที่อยู่ในแถวลำดับคู่ของทรัยส่วนหน้าที่โหนด s ลิงค์ไปหา และจะคืนค่าลิสต์ของโหนดเหล่านี้กลับไป

- ฟังก์ชัน FrChange(s, list)

  ใช้ในการย้ายโหนดต่าง ๆ ที่อยู่ในตัวแปร list ไปยังที่อยู่ใหม่ในแถวลำดับคู่ของทรัยส่วนหน้า เนื่องจากโหนดที่จะเพิ่มใหม่ชนกับโหนดเดิมที่มีอยู่ ซึ่งมีขั้นตอนการทำงานดังนี้

  1. กำหนดค่า oldBase = FrBase[s]
  2. ใช้ฟังก์ชัน FrXCheck(list) หาค่า FrBase[s] ใหม่ที่ทำให้โหนด s สามารถลิงค์ไปหาโหนดทุกโหนดที่อยู่ในตัวแปร list
  3. กำหนดค่า t = oldBase + b (t คือโหนดเก่าที่เกิดการชน และ b คือสมาชิกในตัวแปร list) และกำหนดค่า t' = FrBase[s] + b (t' คือที่อยู่ใหม่ของโหนดเก่า) จากนั้นทำการย้ายที่อยู่ของโหนดเก่าไปยังที่อยู่ใหม่โดยให้ค่า FrBase[t'] = FrBase[t] และ FrCheck[t'] = s
  4. ให้ค่า FrBase[t] = 0 และ FrCheck[t] = 0
  5. ทำงานข้อ 3 และข้อ 4 จนกว่าจะครบสมาชิกทุกตัวในตัวแปร list

- ฟังก์ชัน FrXCheck(list)

เป็นฟังก์ชันที่ผู้วิจัยทำการปรับให้มีประสิทธิภาพดีขึ้นจากงานวิจัยเดิม เพื่อใช้ในการหาค่า FrBase[s] ใหม่ เพื่อทำให้โหนด s สามารถลิงค์ไปยังโหนดทุกโหนดที่อยู่ในตัวแปร list ได้ และไม่ทำให้เกิดการชนกับโหนดอื่น ๆ ที่มีอยู่ในแถวลำดับคู่ของทรัยส่วนหน้า ซึ่งมีขั้นตอนการทำงานดังนี้

1. ใช้สมการ t = Base_Inc + b โดยกำหนดค่าเริ่มต้นของ Base_Inc เท่ากับ 1 และ b เป็นรหัสแอสกีของสมาชิกในตัวแปร list
2. ตรวจสอบค่า FrCheck[t] เท่ากับ 0 หรือไม่ ถ้าเงื่อนไขเป็นจริง ทำงานข้อ 1 โดย b เป็นรหัสแอสกีของสมาชิกตัวถัดไปที่อยู่ในตัวแปร list แต่ถ้าเงื่อนไขเป็นเท็จ ทำงานข้อ 3
3. กำหนดให้ค่า Base_Inc = Base_Inc + 1 จากนั้นเริ่มทำงานตั้งแต่ข้อ 1 โดย b เป็นรหัสแอสกีของสมาชิกตัวแรกในตัวแปร list
4. เมื่อทำงานไปครบทุกสมาชิกในตัวแปร list แล้วจะคืนค่า Base_Inc ซึ่งใช้เป็นค่า FrBase[s] ใหม่กลับไป

- ฟังก์ชัน GetAvailInfIndx()

เป็นฟังก์ชันที่ผู้วิจัยสร้างเพิ่มเติมจากงานวิจัยเดิม เพื่อใช้ในการหาค่าหมายเลขตำแหน่งของอะเรย์ข้อมูลที่ว่างอยู่

- ฟังก์ชัน GetAvailTableIndx()

เป็นฟังก์ชันที่ผู้วิจัยสร้างเพิ่มเติมจากงานวิจัยเดิม เพื่อใช้ในการหาค่าหมายเลขตำแหน่งของอะเรย์เทเบิลที่ว่างอยู่

- ฟังก์ชัน ReAdd(a_{pos+1}..... $a_n a_{n+1}$)

ใช้ในการเพิ่มส่วนที่เหลือของดัชนีในแถวลำดับคู่ของทรัยส่วนหลัง และจะคืนค่าหมายเลขดัชนีของแอคเซพติงโหนดกลับมา ขั้นตอนการทำงานของฟังก์ชันมีดังนี้

1. นำ a_{pos+1}..... $a_n a_{n+1}$ มาทำการผันกลับได้เป็น $a_{n+1} a_n$..... a_{pos+1}
2. ทำการค้นหา $a_{n+1} a_n$..... a_{pos+1} ในแถวลำดับคู่ของทรัยส่วนหลัง โดยใช้สมการ t = ReBase[s] + a_i โดยที่ i = n+1, n, ..., pos+1 ถ้าค้นหาไปจนครบทุกตัวของส่วนที่เหลือของดัชนี ทำงานข้อ 11
3. ตรวจสอบค่า ReCheck[t] เท่ากับ s หรือไม่ ถ้าเท่ากัน ทำงานข้อ 2 จนกว่าค่า ReCheck[t] ไม่เท่ากับ s ให้ทำงานข้อ 4

4. ตรวจสอบค่า ReBase[s] น้อยกว่า 0 หรือไม่ ถ้าเงื่อนไขเป็นจริง ทำงานข้อ 5 แต่ถ้าเงื่อนไขเป็นเท็จ ทำงานข้อ 6
5. กรณีที่ค่า ReBase[s] < 0 แสดงว่าจะต้องเพิ่มโหนดใหม่ต่อจากแอคเซพติงโหนดที่มีอยู่เดิม ต้องสร้างโหนด # เพื่อเป็นตัวปิดท้ายสตริงเดิมในแถวลำดับคู่ของทรัยส่วนหลัง
6. กรณีที่ค่า ReBase[s] > 0 แสดงว่าเป็นการเพิ่มโหนดใหม่ต่อจากโหนดที่มีอยู่เดิม ซึ่งต้องหาค่า ReBase[s] ใหม่เพื่อทำให้โหนด s ที่มีอยู่เดิมสามารถลิงค์มายังโหนดใหม่ด้วย
7. สร้างโหนดใหม่ต่อจากโหนดปัจจุบัน โดยใช้สมการ t = ReBase[s] + a_i
8. ตรวจสอบค่า ReCheck[t] เท่ากับ 0 หรือไม่ ถ้ามีค่าไม่เท่ากับ 0 ทำงานในข้อ 9 ถ้าค่าของ ReCheck[t] เท่ากับ 0 ทำงานในข้อ 10
9. กรณีที่ค่าของ ReCheck[t] ไม่เท่ากับ 0 แสดงว่าเกิดการชนกับโหนดที่มีอยู่เดิม จึงต้องทำการย้ายโหนดโดยเปลี่ยนค่าของ ReBase[s] และ ReBase[t] ใหม่ ซึ่งมีขั้นตอนดังนี้
    - 9.1 ใช้ฟังก์ชัน SetReList(s) เพื่อหาลิสต์ของโหนดต่าง ๆ ที่โหนด s ลิงค์ไปหาซึ่งจะเก็บไว้ใน Slist และใช้ฟังก์ชัน SetReList(ReCheck[t]) เพื่อหา Tlist
    - 9.2 ตรวจสอบจำนวนสมาชิกของ Slist + 1 ว่ามีน้อยกว่าจำนวนสมาชิกของ Tlist หรือไม่ ถ้าเงื่อนไขเป็นจริง ทำงานในข้อ 9.3 แต่ถ้าเงื่อนไขเป็นเท็จ ทำงานในข้อ 9.4
    - 9.3 ใช้ฟังก์ชัน ReChange(s, Slist, a_i) เพื่อย้ายโหนดต่าง ๆ ใน Slist ไปยังที่อยู่ใหม่ที่ไม่ทำให้เกิดการชนกับโหนดใหม่ที่เพิ่มมา โดยจะต้องหาค่า ReBase[s] ใหม่ ซึ่งจะเรียกใช้ฟังก์ชัน ReXCheck(Slist) ในการหาค่า ReBase[s] ที่ทำให้โหนด s สามารถลิงค์ไปหาสมาชิกทุกตัวใน Slist รวมทั้ง a_i ด้วย และต้องอยู่ภายใต้เงื่อนไข ReCheck [ReBase[s]+b] = 0 โดยที่ b เป็นสมาชิกใน Slist U {a_i} จากนั้นทำงานในข้อ 10
    - 9.4 ใช้ฟังก์ชัน ReChange(t, Tlist) เพื่อย้ายโหนดต่าง ๆ ใน Tlist ไปยังที่อยู่ใหม่ที่ไม่ทำให้เกิดการชนกับโหนดใหม่ที่เพิ่มมา โดยจะต้องหาค่า ReBase[t] ใหม่ ซึ่งจะเรียกใช้ฟังก์ชัน ReXCheck(Tlist) ในการหาค่า ReBase[t] ที่ทำให้โหนด t สามารถลิงค์ไปหาสมาชิกทุกตัวใน Tlist และต้องอยู่ภายใต้เงื่อนไข ReCheck [ReBase[t] + b] = 0 โดยที่ b เป็นสมาชิกใน Tlist จากนั้นทำงานในข้อ 10

9. ทำงานตั้งแต่ข้อ 7 ถึงข้อ 9 จนกว่าจะครบทุกตัวของส่วนที่เหลือของดัชนี จากนั้นทำงานข้อ 12
10. กรณีที่ค้นหาจนครบทุกตัวของส่วนที่เหลือของดัชนี แสดงว่าส่วนที่เหลือของดัชนีที่จะทำการเพิ่มมีอยู่ในแถวลำดับคู่ของทรัยส่วนหลังแล้ว ต้องตรวจสอบ a_{pos+1} ตรงกับแอคเซพติงโหนดที่มีอยู่เดิมหรือไม่ ถ้าเงื่อนไขเป็นเท็จ ทำงานข้อ 12 แต่ถ้าเงื่อนไขเป็นจริง ทำงานข้อ 14
11. ตรวจสอบโหนดปัจจุบันที่มีอยู่ในแถวลำดับคู่ของทรัยส่วนหลัง มีลิงค์ไปยังโหนด # ที่เป็นแอคเซพติงโหนดหรือไม่ ถ้าเงื่อนไขเป็นเท็จ ทำงานข้อ 13 แต่ถ้าเงื่อนไขเป็นจริง ทำงานข้อ 14
12. สร้างโหนด # เพื่อเป็นตัวปิดท้ายและเป็นแอคเซพติงโหนดในแถวลำดับคู่ของทรัยส่วนหลังของดัชนีนี้
13. คืนค่าหมายเลขดัชนีของแอคเซพติงโหนดนั้นกลับไป

3.2.2.2 การเพิ่มส่วนของดัชนีที่ไม่พบในแถวลำดับคู่ของทรัยส่วนหลัง

เมื่อทำการค้นหาไปจนถึงเซพพาเรทโหนด และนำส่วนที่เหลือในแถวลำดับคู่ของทรัยส่วนหลังมาเปรียบเทียบกับส่วนที่เหลือของดัชนีแล้ว พบว่าไม่เท่ากัน จึงต้องทำการเพิ่มส่วนของดัชนีที่ไม่พบในแถวลำดับคู่ของทรัยส่วนหลัง ซึ่งการค้นหาในทรัยส่วนหลังของโครงสร้างข้อมูลแบบทู-ทรัยต้องค้นหาในลักษณะผันกลับ นั่นคือทำการค้นหาส่วนที่เหลือของดัชนีจากอักษรตัวสุดท้ายไล่มาจนถึงอักษรตัวแรกแตกต่างจากการค้นหาดัชนีในแถวลำดับคู่ของทรัยส่วนหน้า ซึ่งการเพิ่มส่วนของดัชนีที่ไม่พบในแถวลำดับคู่ของทรัยส่วนหลังใช้ฟังก์ชันต่าง ๆ ดังนี้

กำหนดให้ spn คือเซพพาเรทโหนดที่อยู่ในแถวลำดับคู่ของทรัยส่วนหน้า

$a_{sp\_pos+1}\ldots a_{pos-1}$ คือส่วนที่เหมือนกันของส่วนที่เหลือของดัชนีและส่วนที่เหลือในแถวลำดับคู่ของทรัยส่วนหลัง

$a_{pos}a_{pos+1}\ldots a_n a_{n+1}$ คือส่วนที่เหลือของดัชนีที่แตกต่างจากส่วนที่เหลือในแถวลำดับคู่ของทรัยส่วนหลัง

r คือพาเรนท์โหนดของ $a_{sp\_pos+1}\ldots a_{pos-1}$ ที่อยู่ในแถวลำดับคู่ของทรัยส่วนหลัง

- ฟังก์ชัน Reinsert(spn, r, $a_{sp\_pos+1}\ldots a_{pos-1}$, $a_{pos}a_{pos+1}\ldots a_n a_{n+1}$)

นำ $a_{sp\_pos+1}\ldots a_{pos-1}$ เพิ่มต่อจาก spn ที่อยู่ในแถวลำดับคู่ของทรัยส่วนหน้า จากนั้นทำการเพิ่มโหนด r และ a_{pos} ต่อจาก a_{pos-1} ในแถวลำดับคู่ของทรัยส่วนหน้า และเพิ่ม $a_{pos+1}\ldots a_n a_{n+1}$ ใน

แถวลำดับคู่ของทรัยส่วนหลัง ซึ่งมีขั้นตอนการทำงานของฟังก์ชันดังนี้

1. กำหนดค่า oldSpn = FrBase[spn]
2. ใช้ฟังก์ชัน FrXCheck(a_{sp_pos+1}) เพื่อหาค่า FrBase[spn]
3. กำหนดค่า s = spn
4. นำ $a_{sp\_pos+1}...a_{pos-1}$ เพิ่มต่อจาก spn โดยใช้สมการ t = FrBase[s] + a_i โดยที่ i = sp_pos+1, ..., pos-1 ถ้าเป็นการเพิ่มอักษรถัดไปต้องให้ค่าเริ่มต้น s = t เดิม
5. ใช้ฟังก์ชัน FrXCheck(a_{pos} r) เพื่อหาค่า FrBase[a_{pos-1}]
6. เพิ่มโหนด r ต่อจาก a_{pos-1} ในแถวลำดับคู่ของทรัยส่วนหน้าโดยใช้สมการ t = FrBase[s] + r
7. กำหนดค่า FrBase[r] = oldSpn
8. ใช้ฟังก์ชัน GetAvailInfIndex() เพื่อหาหมายเลขตำแหน่งของอะเรย์ข้อมูลที่ว่างและกำหนดหมายเลขดัชนีนั้นให้กับ FrBase[a_{pos}]
9. ใช้ฟังก์ชัน FrInsert(a_{pos-1}, $a_{pos}a_{pos+1}...a_n a_{n+1}$) เพื่อทำการเพิ่ม $a_{pos}a_{pos+1}...a_n a_{n+1}$ ในโครงสร้างข้อมูลแบบทู-ทรัย
10. สร้างโหนด # เพิ่มต่อจากพาเรนท์โหนดของโหนด r เพื่อใช้ปิดท้าย $a_{sp_pos+1}...a_{pos-1}$r ที่อยู่ในแถวลำดับคู่ของทรัยส่วนหลังและเป็นแอคเซพติงโหนด
11. ใช้ฟังก์ชัน DelSome(acc) เพื่อลบ $a_{sp\_pos+1}...a_{pos-1}$ ที่อยู่ในแถวลำดับคู่ของทรัยส่วนหลัง โดย acc คือหมายเลขดัชนีของแอคเซพติงโหนด

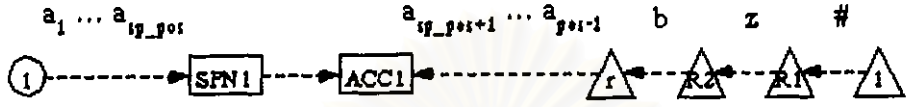
การทำงานของฟังก์ชัน ReInsert(spn, r, $a_{sp\_pos+1}...a_{pos-1}$, $a_{pos}a_{pos+1}...a_n a_{n+1}$) แสดงดังรูปที่ 3.5

- ฟังก์ชัน DelSome(acc) โดย acc คือหมายเลขดัชนีของแอคเซพติงโหนดที่อยู่ในแถวลำดับคู่ของทรัยส่วนหลัง

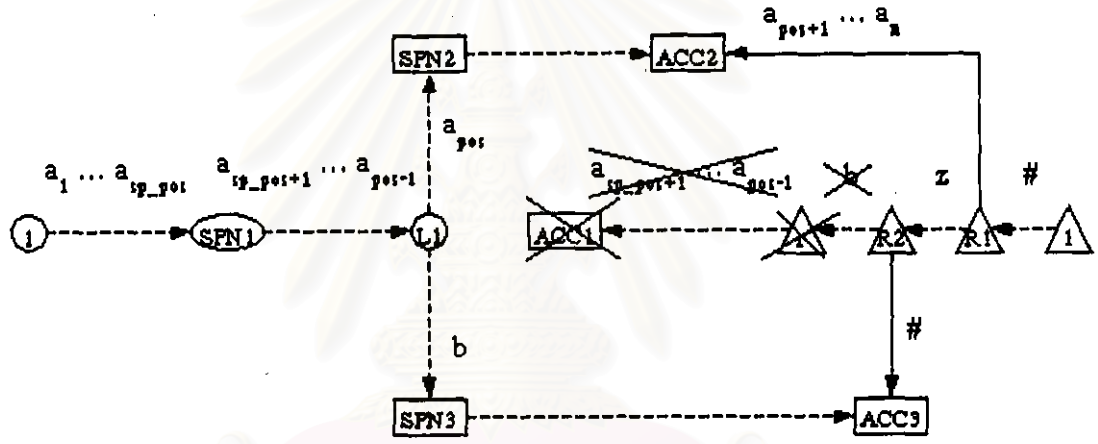
ใช้ในการลบ $a_{sp\_pos+1}...a_{pos-1}$ ที่อยู่ในแถวลำดับคู่ของทรัยส่วนหลังเพื่อขจัดโหนดที่ไม่ได้ใช้งานออกไป โดยต้องอยู่ภายใต้เงื่อนไขคือค่า TableCnt[m] ต้องมีค่าน้อยกว่าหรือเท่ากับ 1 โดย m คือหมายเลขตำแหน่งของอะเรย์เทเบิลที่ได้มาจาก ReBase[acc] คูณด้วย -1

สำหรับฟังก์ชัน SetReList(s) ฟังก์ชัน ReChange(s, list) และฟังก์ชัน ReXCheck(list) คล้ายกับฟังก์ชัน SetFrList(s) ฟังก์ชัน FrChange(s, list) และฟังก์ชัน FrXCheck(list) ตามลำดับ เพียงแต่กระทำบนแถวลำดับคู่ของทรัยส่วนหลัง

หลักการของขั้นตอนวิธีเพิ่มดัชนีคือเมื่อเพิ่มตัวอักษรหรือโหนดใหม่เข้าไปและถ้าตำแหน่งที่โหนดใหม่ต้องการถูกใช้โดยโหนดอื่น จะต้องทำการย้ายโหนดใดโหนดหนึ่งออกไปและหาตำแหน่งใหม่ที่ว่างสำหรับโหนดทุกโหนดที่สัมพันธ์กับโหนดนั้น ดังนั้นหลังจากที่ทำการเพิ่มดัชนีใหม่ในโครงสร้างแถวลำดับคู่ จะทำให้ตำแหน่งที่อยู่ของโหนดต่าง ๆ เปลี่ยนไปเรื่อย ๆ แต่จะพบว่าตำแหน่งแรกของแถวลำดับคู่ไม่เปลี่ยนเสมอ



(ก) ทรัยส่วนหลัง ก่อนทำการเพิ่มดัชนี



(ข) ทรัยส่วนหลัง หลังทำการเพิ่มดัชนี

รูปที่ 3.5 การทำงานของฟังก์ชัน ReInsert

เวลาที่ใช้ในการเพิ่มดัชนี 1 คำในโครงสร้างแถวลำดับคู่คำนวณมาจากขั้นตอนวิธีการเพิ่มดัชนีโดยกำหนดให้ k คือจำนวนตัวอักษรที่ประกอบในดัชนี h คือลิสต์ของตำแหน่งช่องว่างในแถวลำดับคู่และ m คือจำนวนชนิดของตัวอักษร(The kind of input characters) เมื่อทำการเพิ่มโหนดใหม่และโหนดใหม่ชนกับโหนดเดิมที่มีอยู่ ขั้นตอนวิธีการเพิ่มดัชนีนี้จะทำการหาค่าแบสที่เหมาะสมให้กับโหนดใหม่ซึ่งเวลาที่ใช้ในการหาค่าแบสใหม่มากที่สุดเท่ากับ m ครั้งในแต่ละครั้งจะต้องทำการตรวจสอบว่าตำแหน่งที่อยู่ของโหนดใหม่ที่ได้มาจากการหาค่าแบสใหม่นั้นเป็นตำแหน่งที่อยู่ในลิสต์ h หรือไม่ดังนั้นการเพิ่มตัวอักษรหรือโหนดใหม่ 1 โหนดใช้เวลา O(hm) และเวลาที่ใช้ในการเพิ่มดัชนี 1 คำที่ประกอบด้วย k ตัวอักษร[5] เท่ากับ O(khm)

---

[5] Ibid., p. 487.

3.2.3 ขั้นตอนวิธีลบดัชนีและข้อมูลของดัชนี

การลบดัชนีและข้อมูลของดัชนีภายในโครงสร้างข้อมูลแบบทู-ทรัยจะต้องทำการค้นหาดัชนีก่อน เมื่อพบแล้วจึงทำการลบดัชนีและข้อมูลของดัชนีนั้นออกไป

การลบดัชนีในโครงสร้างข้อมูลแบบทู-ทรัยของเครื่องมือที่พัฒนาขึ้นแบ่งเป็น 2 แบบคือ การลบดัชนี และการลบเฉพาะข้อมูลของดัชนีที่มีอยู่เดิมเนื่องจากดัชนีที่ใช้เป็นแอตทริบิวท์ที่ไม่ใช่คีย์หลักจึงทำให้มีข้อมูลของดัชนีหนึ่ง ๆ มากกว่า 1 ข้อมูลซึ่งในงานวิจัยนี้ได้ทำการจัดเก็บข้อมูลของแต่ละดัชนีในลักษณะเป็นลิงค์ลิสต์ การลบดัชนีจะทำได้ต่อเมื่อดัชนีนั้นมีจำนวนข้อมูลเพียง 1 ข้อมูลและข้อมูลนั้นตรงกับข้อมูลของดัชนีที่ระบุ ส่วนการลบเฉพาะข้อมูลของดัชนีจะใช้ในกรณีที่พบว่าดัชนีนั้นมีจำนวนข้อมูลมากกว่า 1 ข้อมูลซึ่งการลบดัชนีและข้อมูลของดัชนีใช้ฟังก์ชันดังนี้

กำหนดให้ spn คือเซพพาเรทโหนดที่อยู่ในแถวลำดับคู่ของทรัยส่วนหน้า

- ฟังก์ชัน DeleteKey(spn)

เป็นฟังก์ชันที่ผู้วิจัยปรับการทำงานของฟังก์ชันจากงานวิจัยเดิมเพื่อให้ทำงานสอดคล้องกับส่วนของการจัดเก็บข้อมูลของแต่ละดัชนีที่ได้ออกแบบไว้ ฟังก์ชันนี้จะทำการลบเซพพาเรทโหนดที่อยู่ในแถวลำดับคู่ของทรัยส่วนหน้า และส่วนที่เหลือที่อยู่ในแถวลำดับคู่ของทรัยส่วนหลังของดัชนีที่ระบุ ซึ่งขั้นตอนการทำงานของฟังก์ชันมีดังนี้

1. กำหนดค่า n = FrBase[spn] * (-1)
2. กำหนดค่า headPtr = InfPtr[n]
3. กำหนดค่า curr = headPtr -> next
4. ไล่ไปตามลิงค์ลิสต์ของข้อมูลโดยกำหนดให้ curr = curr -> next และเปรียบเทียบว่า curr -> info ของโหนดใดตรงกับข้อมูลของดัชนีที่ระบุว่าต้องการลบ ถ้าไม่พบข้อมูลที่ระบุ ทำงานข้อ 5 แต่ถ้าพบ ทำงานข้อ 6
5. ระบุว่าไม่พบข้อมูลและออกจากฟังก์ชัน
6. ปลดปล่อย (Free) โหนดที่มีข้อมูลของดัชนีตรงกับที่ระบุ
7. กำหนดค่า headPtr -> cnt = headPtr -> cnt - 1 เป็นการลดจำนวนข้อมูลของดัชนีนั้นลงไปหนึ่งจำนวน
8. ตรวจสอบค่า headPtr -> cnt เท่ากับ 0 หรือไม่ ถ้าเงื่อนไขเป็นเท็จ ทำงานข้อ 9 แต่ถ้าเงื่อนไขเป็นจริง ทำงานข้อ 9

9. ไม่ทำการลบดัชนีและออกจากฟังก์ชัน
10. ถ้า headPtr -> cnt = 0 จะทำการลบดัชนีออกไปด้วย โดยกำหนดค่า FrBase[spn] = 0 และค่า FrCheck[spn] = 0
11. ใช้ฟังก์ชัน GetLinkState(spn) เพื่อหาหมายเลขดัชนีของแอคเซพติงโหนดที่อยู่ในแถวลำดับคู่ของทรัยส่วนหลัง ซึ่งกำหนดให้เป็นค่า acc
12. ใช้ฟังก์ชัน DelSome(acc) เพื่อทำการลบส่วนที่เหลือของดัชนีที่อยู่ในแถวลำดับคู่ของทรัยส่วนหลัง

- ฟังก์ชัน GetLinkState(spn)

ใช้ในการหาหมายเลขดัชนีของแอคเซพติงโหนดที่อยู่ในแถวลำดับคู่ของทรัยส่วนหลัง ซึ่งเชื่อมกับ spn ที่อยู่ในแถวลำดับคู่ของทรัยส่วนหน้า

เวลาที่ใช้ในการลบดัชนี 1 คำคำนวณได้จากขั้นตอนวิธีลบดัชนีโดยกำหนดให้ k คือจำนวนตัวอักษรที่ประกอบในดัชนีและ m คือจำนวนชนิดของตัวอักษร เวลาที่ใช้ในการลบส่วนที่เหลือของดัชนีในแถวลำดับคู่ของทรัยส่วนหลังมากที่สุดเท่ากับ O(k) และเวลาที่ใช้ในการลบแอคเซพติงโหนดที่เป็นโหนด # เท่ากับ O(m) ดังนั้นเวลาที่ใช้ในการลบดัชนี 1 คำโดยประมาณ[6] เท่ากับ O(k+m)

3.3 การปรับปรุงโครงสร้างข้อมูลของดัชนี

เมื่อทำการพัฒนาขั้นตอนวิธีเพิ่มดัชนีและข้อมูลของดัชนีของโครงสร้างข้อมูลทู-ทรัยแบบโครงสร้างแถวลำดับคู่โดยใช้ภาษาซีแล้วจึงได้ทำการทดสอบขั้นตอนวิธีดังกล่าว โดยการทดสอบจะทำการวัดขนาดดัชนีที่สร้างขึ้นในหน่วยความจำใช้หน่วยไบต์ (Byte) จำนวนครั้งที่เกิดการชนต่อการเพิ่มดัชนี 1 ระเบียนและเวลาที่ใช้ในการสร้างดัชนีทั้งหมด ส่วนผลการทดสอบขั้นตอนวิธีค้นหาข้อมูลและขั้นตอนวิธีลบดัชนีและข้อมูลของดัชนีจะกล่าวไว้ในบทที่ 5 ในหัวข้อ 5.1

ตารางผลการทดสอบขั้นตอนวิธีการเพิ่มดัชนีและข้อมูลของดัชนีจะแสดงไว้ในบทที่ 5 ในหัวข้อ 5.1.1 ซึ่งจากผลการทดสอบเพิ่มข้อมูลรายชื่อนิสิตภาษาไทยและข้อมูลรายชื่อนิสิตภาษาอังกฤษในโครงสร้างข้อมูลทู-ทรัยแบบโครงสร้างแถวลำดับคู่พบว่า ขนาดของแถวลำดับคู่มีขนาดเล็ก (Compact) และมีจำนวนช่องว่างน้อยทำให้ใช้เนื้อที่ในการสร้างดัชนีมีประสิทธิภาพดี แต่

---

[6] Ibid., p. 487.

จำนวนครั้งที่ตัวอักษรหรือโหนดใหม่ที่เพิ่มเข้าไปชน (Collsion) กับโหนดเดิมที่มีอยู่มีเป็นจำนวนมากทำให้เสียเวลาในการย้ายโหนดที่ถูกชนไปยังที่อยู่ใหม่ซึ่งส่งผลให้เวลาที่ใช้ในการสร้างดัชนีเพิ่มขึ้น

การชนจะเกิดในขั้นตอนการทำงานของฟังก์ชัน FrXCheck(list) และฟังก์ชัน ReXCheck(list) ที่ใช้ในการหาค่า FrBase[s] และ ReBase[s] ใหม่ ขั้นตอนวิธีของฟังก์ชัน FrXCheck(list) และ ReXCheck(list) คล้ายกัน ฟังก์ชันทั้ง 2 แตกต่างกันตรงที่ฟังก์ชัน FrXCheck(list) ใช้สำหรับ ทรัยส่วนหน้า ส่วนฟังก์ชัน ReXCheck(list) ใช้สำหรับทรัยส่วนหลัง ตัวอย่างขั้นตอนวิธีของฟังก์ชัน FrXCheck(list) แสดงดังรูปที่ 3.6

```
int FrXCheck(list)    {
    do  {
        t = Base_Inc + a_h;
        if (FrCheck[t] != 0)   {
            Base_Inc = Base_Inc + 1;
            h = 0;   /* loop start at the first char */
        }
        else
            h = h + 1;    /* next char in list */
    } while (h == n);      /* end of list */
    return(Base_Inc);
}
```

รูปที่ 3.6 ขั้นตอนวิธีของฟังก์ชัน FrXCheck

จากรูปที่ 3.6 กำหนดให้ a_h คือรหัสแอสกีของตัวอักษรแต่ละตัวที่อยู่ในตัวแปร list โดย h แทนตำแหน่งของตัวอักษรในตัวแปร list เริ่มจากตำแหน่ง 0, 1, ..., n-1 (n คือจำนวนตัวอักษรทั้งหมดที่อยู่ในตัวแปร list) ส่วน t คือตำแหน่งที่จะให้โหนด a_h อยู่ และฟังก์ชัน FrXCheck(list) กำหนดให้ค่าเริ่มต้นของ Base_Inc เท่ากับ 1

ขั้นตอนการทำงานของฟังก์ชัน FrXCheck(list) เริ่มจากหาค่าตำแหน่งที่โหนด a_h จะย้ายไปอยู่ (ค่า t) จากการนำรหัสแอสกีของตัวอักษรใน list (a_h) บวกด้วยค่า Base_Inc จากนั้นตรวจสอบว่าตำแหน่งที่โหนด a_h จะไปอยู่ว่างหรือไม่ ถ้าว่าง (FrCheck[t] = 0) จะใช้ค่า Base_Inc เดิมในการคำนวณหาตำแหน่งที่โหนดถัดไปที่อยู่ในตัวแปร list จะไปอยู่ แต่ถ้าตำแหน่งที่โหนดจะไปอยู่ไม่ว่าง (FrCheck[t] != 0) แสดงว่าตำแหน่งที่โหนด a_h จะไปอยู่มีโหนดอื่นอยู่ที่ตำแหน่งนั้นแล้ว เมื่อโหนดที่เพิ่มใหม่ชนกับโหนดที่มีอยู่เดิม ฟังก์ชันจะทำการเพิ่มค่า Base_Inc ทีละ I ไปจนกระทั่งค่า FRCheck[t] ว่าง ซึ่งถ้ามีจำนวนดัชนีที่จะเพิ่มในโครงสร้างข้อมูลแบบทู-ทรีมากขึ้นจะทำให้เกิดการชนมากขึ้นและส่งผลให้เวลาในการสร้างดัชนีเพิ่มขึ้นด้วย

ผู้วิจัยจึงได้ทำการปรับขนาดของ Base_Inc ให้เหมาะสมโดยการหาค่า Base_Inc ของแต่ละตัวอักษรเพื่อที่จะทำให้การชนลดลงและใช้เวลาในการสร้างดัชนีน้อยลง ซึ่งขั้นตอนวิธีการหาค่า Base_Inc มีรายละเอียดดังนี้

1. ทดสอบค่า Base_Inc แต่ละค่าตั้งแต่ค่า 1 ถึง 30 โดยใช้ข้อมูลรายชื่อภาษาไทยและรายชื่อภาษาอังกฤษของนิสิตจำนวน 25,000 ระเบียนในการทดสอบขั้นตอนวิธีการเพิ่มดัชนี
2. ทำการวัดขนาดเนื้อที่ที่ใช้สร้างโครงสร้างดัชนีในหน่วยไบต์ จำนวนครั้งที่เกิดการชนและเวลาที่ใช้ในการสร้างดัชนีทั้งหมดในหน่วยวินาที
3. คำนวณหาผลคูณระหว่างขนาดของโครงสร้างดัชนีและเวลาที่ใช้ในการสร้างดัชนีทั้งหมดของแต่ละค่า Base_Inc ซึ่งผลการทดสอบค่า Base_Inc ตั้งแต่ค่า 1 ถึง 30 จะแสดงไว้ในตารางที่ 5.3 และ 5.4 หัวข้อ 5.1.2 ของบทที่ 5
4. เรียงลำดับผลคูณนั้นจากน้อยไปมาก ซึ่งค่า Base_Inc ที่มีผลคูณน้อยที่สุดจะเป็นค่า Base_Inc ที่ดีที่สุด (Optimum) เนื่องจากค่า Base_Inc นั้นทำให้ใช้เนื้อที่สร้างโครงสร้างดัชนีน้อยและใช้เวลาในการสร้างดัชนีน้อย ซึ่งถ้าใช้เวลาในการสร้างดัชนีน้อยจะหมายถึงค่า Base_Inc นั้นทำให้จำนวนครั้งที่เกิดการชนน้อยด้วย
5. นับความถี่ที่ใช้ของแต่ละตัวอักษรในข้อมูลที่ใช้ในการทดสอบ จากนั้นเรียงตัวอักษรที่มีความถี่จากมากไปน้อย
6. จัดให้ตัวอักษรที่มีความถี่ใกล้เคียงกันให้อยู่กลุ่มเดียวกัน โดยให้กลุ่มตัวอักษรที่มีความถี่มากใช้ค่า Base_Inc ที่มีค่าผลคูณน้อย เนื่องจากสมมติฐานว่าตัวอักษรที่มีความถี่มากจะทำให้เกิดการชนมาก จึงได้จัดให้กลุ่มตัวอักษรที่ทำให้เกิดการชนมากใช้ค่า Base_Inc ที่มีค่าผลคูณน้อย และจัดค่า Base_Inc ให้เรียงไปตามลำดับความถี่ของแต่ละกลุ่มตัวอักษร

จากการทดลองตามขั้นตอนวิธีการหาค่า Base_Inc ที่ออกแบบไว้ทำให้ได้ค่า Base_Inc ของแต่ละตัวอักษรภาษาอังกฤษและภาษาไทยซึ่งแสดงไว้ในตารางที่ 5.5 และ 5.6 ตามลำดับในหัวข้อ 5.1.2 ของบทที่ 5 จากนั้นผู้วิจัยได้ทำการปรับปรุงขั้นตอนวิธีของฟังก์ชัน FrXCheck(list) และ ReXCheck(list) บางส่วนซึ่งแสดงไว้ดังรูปที่ 3.7 เพื่อทำให้ได้ค่า Base_Inc ที่เหมาะสมและช่วยลดจำนวนครั้งที่เกิดการชน

```
1: int FrXcheck(list)  {
2:     do  {
3:         t = Base_Inc + a_h;
4:         if (FrCheck[t] != 0)  {
(5):           Base_Inc = Base_Inc + BaseChar[a_h];
6:             h = 0;   /* loop start at the first char */
7:         }
8:         else
9:             h = h + 1;    /* next char in list */
10:    } while (h == n);      /* end of list */
11:    return(Base_Inc);
12: }
```

รูปที่ 3.7 การปรับปรุงขั้นตอนวิธีของฟังก์ชัน FrXCheck

จากรูปที่ 3.7 ผู้วิจัยได้ทำการปรับปรุงขั้นตอนวิธีของฟังก์ชัน FrXCheck(list) ในบรรทัดที่ 5 จากเดิม Base_Inc = Base_Inc + 1; มาเป็น Base_Inc = Base_Inc + BaseChar[a_h]; โดยที่ BaseChar[a_h] คือค่าเบสของตัวอักษร a_h ที่ได้จากการปรับขนาดค่าเบสที่ได้แสดงไว้ในตารางที่ 5.5 และ 5.6 ในหัวข้อ 5.1.2 ของบทที่ 5

หลังจากปรับปรุงขั้นตอนวิธีของฟังก์ชัน FrXCheck(list) และ ReXCheck(list) บางส่วนแล้วจึงได้ทำการทดสอบขั้นตอนวิธีเพิ่มดัชนีอีกครั้งโดยใช้ข้อมูลรายชื่อภาษาไทยและข้อมูลรายชื่อภาษาอังกฤษซึ่งเป็นชุดเดียวกับที่ใช้ในการทดสอบก่อนปรับขนาดของค่าเบส โดยการทดสอบครั้งนี้จะใช้ค่า Base_Inc ที่ได้ทำการปรับปรุงแล้ว ซึ่งตารางผลการทดสอบและกราฟแสดงการ

เปรียบเทียบผลการทดสอบก่อนและหลังปรับขนาดของค่าเบสจะแสดงไว้ในบทที่ 5 ในหัวข้อ 5.1.3

เมื่อได้โครงสร้างข้อมูลของดัชนีที่มีประสิทธิภาพสำหรับใช้ในการค้นหา เพิ่มและลบดัชนีแล้ว ในบทถัดไปจะกล่าวถึงการออกแบบและพัฒนาเครื่องมือซึ่งจะนำโครงสร้างข้อมูลทูพิลทรัยแบบโครงสร้างแถวลำดับคู่นี้ไปใช้เป็นโครงสร้างข้อมูลของดัชนี