การตัดคำแบบปลอดภัยโดยใช้พจนานุกรมสำหรับทำดัชนีฐานข้อมูลเอกสารภาษาไทย

Thai Word Safe Segmentation for Data Indexing in Dictionary-based Search Engine

อัจฉริยา กล้าหาญ[1] สุกัญญา พันธ์น้อย[1] ประพีร์พัฒน์ เอื้อวิจิตรพจนา[2] และ รุ่งรัตน์ เวียงศรีพนาวัลย์[1]

[1]ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
[2]บริษัท ทิงค์เน็ต จำกัด กรุงเทพมหานคร
Emails: achariyakl@gmail.com, pnsukunya@gmail.com, prapeepat@thinknet.co.th, kwrungrat@gmail.com

บทคัดย่อ

ปัญหาคำกำกวมในการตัดคำภาษาไทยโดยใช้พจนานุกรมส่งผลให้ดัชนีฐานข้อมูลในการค้นหาคำไม่ครบถ้วนถูกต้อง บทความนี้เสนอวิธีการแก้ปัญหาคำกำกวมด้วยการตัดคำแบบปลอดภัยโดยใช้พจนานุกรม ส่งผลให้ดัชนีฐานข้อมูลประกอบด้วยคำทุกรูปแบบในบริบทที่กำกวมและเสนอวิธีการตัดคำกรณีมีคำที่สะกดผิดด้วยวิธีการตัดคำแบบไม่ปลอดภัย ในบทความมีการเปรียบเทียบประสิทธิภาพของโครงสร้างทรัยที่เป็นโครงสร้างหลักในการตัดคำโดยวัดเวลาการทำงานเพื่อหาโครงสร้างทรัยที่ดีที่สุดและมีการเปรียบเทียบประสิทธิภาพของไลบรารีในการแปลงสายอักขระให้เป็นข้อมูลเชิงวัตถุเมื่อใช้โครงสร้างทรัยในการกำหนดค่าตั้งต้นของโมดูลการตัดคำ ผลการทดลองพบว่าโครงสร้างทรัยแบบลิ้งค์ลิสต์และไลบรารีโปรโตสตัฟฟ์มีประสิทธิภาพดีที่สุด โมดูลการตัดคำแบบปลอดภัยให้ผลลัพธ์เป็นคำทุกคำที่เป็นไปได้แต่ใช้เวลาทำงานนานและไม่มีคำตอบเมื่อไม่พบคำในพจนานุกรม โมดูลการตัดคำแบบไม่ปลอดภัยให้คำตอบรวดเร็วเสมอในทุกกรณีแต่มีคำตอบเดียว

คำสำคัญ: การตัดคำภาษาไทยโดยใช้พจนานุกรม; การแปลงสายอักขระให้เป็นวัตถุ; โครงสร้างทรัย; ดัชนีฐานข้อมูล

ABSTRACT

Word segmentation ambiguity in Thai language affects data indexing process by creating the inverted index relatively to the segmentation result. This phenomenon leads to unreasonable search result. The article proposes Thai word Safe segmentation algorithm using dictionary to solve this problem so that all different terms in ambiguous part of the sentence are queryable. Next, it provides Thai word Unsafe segmentation algorithm to solve the word-misspelled sentences. The article also compares several off-the-shelf implementations of the Trie data structure which -we believe- is the best data structure for dictionary-based Thai word segmentation and compares the efficiency of libraries for de-serializing values in the segmentation modules' initial state. Finally, it evaluates the Safe and Unsafe segmentation modules called SafeAnalyzer and UnsafeAnalyzer. The experimental results show that the linked-list Trie and Protostuff library give the best results. The Safe segmentation can definitely solve the ambiguity problem but consumes segmenting time searching for ambiguity terms and suffers the unregistered word case. The Unsafe segmentation always provides the rapid but only one answer.

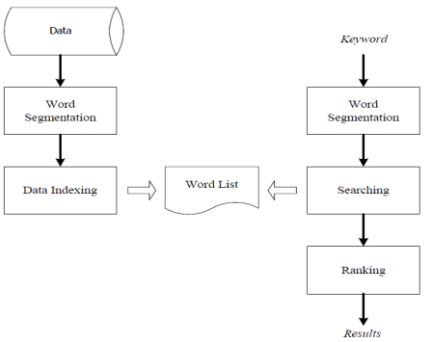
Keyword: Dictionary- based Thai word segmentation, De-serialization, Trie structure, index database

1. บทนำ

ระบบการค้นหาข้อมูลหรือเสิร์ชเอ็นจิน เป็นที่นิยมกันอย่างมากในระบบเครือข่ายอินเทอร์เน็ต เช่น กูเกิลและยาฮู โดยปกติแล้วระบบการค้นหาข้อมูลจะทำการค้นหาข้อมูลผ่านการทำดัชนีฐานข้อมูล

การทำดัชนีฐานข้อมูลในแต่ละภาษามีกระบวนการตัดคำที่แตกต่างกันไป สำหรับภาษาไทยมีกระบวนการตัดคำที่แตกต่างกันอยู่ 2 กลุ่มคือ การตัดคำที่ใช้พจนานุกรม [1, 2] และการตัดคำที่ไม่ใช้พจนานุกรม [3, 4] ภายในบทความนี้สนใจในส่วนของการตัดคำที่ใช้พจนานุกรม

กระบวนการทำงานของระบบการค้นหาข้อมูลภาษาไทยโดยใช้พจนานุกรม จะนำข้อมูลที่เป็นเอกสารมาเข้ากระบวนการการตัดคำให้เป็นรายการคำย่อย ซึ่งใช้เป็นตัวแทนในการค้นหาข้อมูล เมื่อต้องการค้นหาจะทำการระบุคำค้นหา และนำเข้ากระบวนการตัดคำให้เป็นรายการคำย่อยโดยใช้พจนานุกรมเดียวกัน จากนั้นนำไปเทียบเคียงกับรายการคำย่อยของแต่ละเอกสาร ผลลัพธ์เอกสารจะมีการจัดอันดับที่เป็นไปตามรูปแบบที่กำหนดไว้ล่วงหน้า แสดงการทำงานดังรูปที่ 1



รูปที่ 1. ระบบการค้นหาโดยใช้พจนานุกรม

ความถูกต้องครบถ้วนของการตัดคำส่งผลกระทบโดยตรงกับความแม่นยำของการค้นหา Sornlertlamvanich และคณะ [3] แบ่งข้อผิดพลาดของการตัดคำออกเป็น 2 กรณี

<u>กรณีที่ 1 คำกำกวม</u> เนื้อหาของเอกสาร A คือ abcbcb โดยใช้โมดูลการตัดคำ เนื้อหาจะแบ่งเป็น a | bc | bc | b (สมมุติว่าการตัดคำที่ถูกต้องคือ a | b | cb | cb) กรณีนี้ถ้า query cb จะไม่สามารถพบได้ในเอกสาร A หรือถ้า query bc เอกสาร A จะถูกส่งกลับมาไม่ถูกต้อง ตัวอย่างคำกำกวม เช่นคำว่า ตากลม ซึ่งสามารถอ่านได้เป็นสองแบบ คือ ตาก-ลม และ ตา-กลม โมดูลการตัดคำจะให้คำตอบเดียว ดังนั้นถ้าผลลัพธ์ของโมดูลการตัดคำเป็น ตาก-ลม จะไม่พบคำว่า ตา-กลม ในเอกสาร

<u>กรณีที่ 2 คำที่ไม่พบในพจนานุกรม</u> เนื้อหาของเอกสาร A คือ abcdac โดยใช้โมดูลการตัดคำ เนื้อหาจะแบ่งเป็น a | bc | d |ac (สมมติว่าการตัดคำที่ถูกต้องคือ a | b | cd |ac และ cd เป็น unregistered word จากการตัดคำ) ถ้า query cd ผลลัพธ์จากเอกสารฉบับนี้จะไม่ถูกต้อง เนื่องจากไม่พบ cd ในเอกสาร A

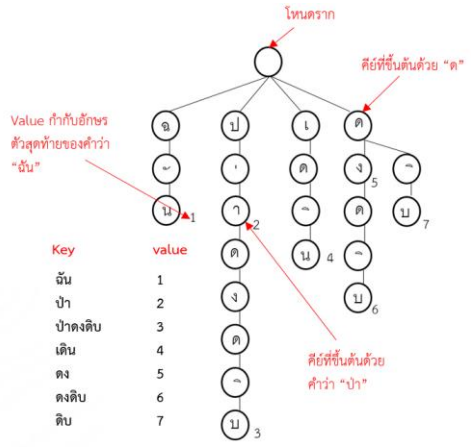
ดังนั้น ความผิดพลาดของการตัดคำจะส่งผลต่อผลลัพธ์ของการค้นหา บทความนี้จึงเสนอวิธีแก้ปัญหาคำกำกวมในการตัดคำด้วยขั้นตอนวิธีใหม่ที่ชื่อว่า Safe segmentation ซึ่งใช้โครงสร้างทรัยในการสนับสนุนการตัดคำเพื่อให้ผลลัพธ์เป็นทุกคำที่เป็นไปได้ในคำกำกวมและขั้นตอนวิธี Unsafe segmentation สำหรับประโยคที่มีคำสะกดผิด

2. ทฤษฎีที่เกี่ยวข้อง

ทฤษฎีที่เกี่ยวข้องกับการตัดคำอย่างปลอดภัยโดยใช้พจนานุกรมสำหรับทำดัชนีฐานข้อมูลเอกสารภาษาไทย มีรายละเอียดดังต่อไปนี้

2.1 โครงสร้างทรัย (Trie structure)

โครงสร้างทรัย [10] หรือ prefix tree เป็นทรีแบบลำดับที่เก็บความสัมพันธ์ด้วยคีย์และเก็บข้อมูลโดยใช้โครงสร้างที่เป็นข้อความ รูปแบบการเพิ่มข้อมูลในทรัยจะใช้คุณสมบัติในการสร้างความสัมพันธ์ของคีย์เพื่ออ้างอิงไปยังโหนดถัดไป ซึ่งโครงสร้างทรัยเป็นโครงสร้างข้อมูลที่นิยมใช้สำหรับเก็บพจนานุกรม แสดงตัวอย่างดังรูปที่ 2



รูปที่ 2. โครงสร้างทรัย

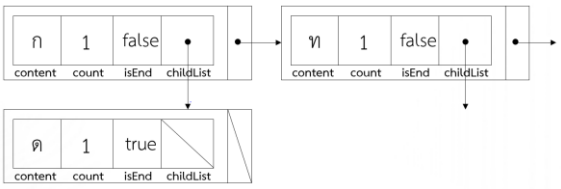
ขั้นตอนการตัดคำจะต้องมีการค้นหาคำในรายการคำ โดยใช้โครงสร้างทรัยเป็นตัวสืบค้น รูปแบบของโครงสร้างทรัยมีลักษณะเด่นในการค้นหาคำที่มีคำขึ้นต้นเหมือนกัน ทำให้การค้นหาคำเกิดขึ้นได้อย่างรวดเร็ว

2.2 Trie implementation

บทความนี้ได้ศึกษาค้นคว้าและวิเคราะห์โครงสร้างทรัย เพื่อทำการคัดเลือกมาใช้ในการพัฒนามีทั้งหมด 4 โครงสร้าง โดยในที่นี้เรียกว่า TrieA [7], TrieB [8], TrieC [9] และ TrieD [10] ตามลำดับ ซึ่งทั้ง 4 โครงสร้างมีทั้งหมด 3 รูปแบบ ดังนี้

2.2.1 TrieA มีโครงสร้างข้อมูลแบบลิ้งค์ลิสต์ (Linklist)

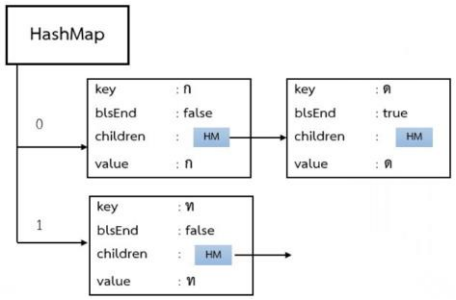
โครงสร้างทรัยที่มีโครงสร้างการทำงานแบบลิ้งค์ลิสต์ ซึ่งลิ้งค์ลิสต์มีคุณสมบัติเรียงลำดับข้อมูลภายในลิสต์ที่มีลักษณะเป็นลำดับต่อเนื่อง โดยอักขระแต่ละตัวจะเชื่อมโยงกับตัวถัดไปในลักษณะรายการต่อเนื่องกันไป โครงสร้างภายในทรัยแสดงตัวอย่างในรูปที่ 3



รูปที่ 3. TrieNode ที่มีโครงสร้างการทำงานแบบลิ้งค์ลิสต์

2.2.2 TrieB และ TrieC มีโครงสร้างข้อมูลแบบ HashMap

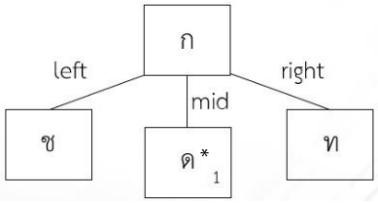
โครงสร้างทรัยที่มีโครงสร้างการทำงานแบบ HashMap โดย HashMap จะเป็นรูปแบบการจัดเก็บค่าตัวแปร ที่อยู่ในรูปแบบของ Key และ Value โครงสร้างภายใน TrieNode จะแสดงตัวอย่างดังรูปที่ 4



รูปที่ 4. TrieNode ที่มีโครงสร้างการทำงานแบบ HashMap

2.2.3 TrieD มีโครงสร้างข้อมูลแบบ Node tree

โครงสร้างทรัยที่มีโครงสร้างการทำงานแบบ Node tree จะมีการเก็บค่าที่สัมพันธ์กับตัวอักษรตัวสุดท้ายของแต่ละคำไว้ภายในโหนด โครงสร้างภายใน TrieNode จะแสดงตัวอย่างดังรูปที่ 5



รูปที่ 5. TrieNode ที่มีโครงสร้างการทำงานแบบ Node tree

*1 คือ ค่า Value ที่เกี่ยวข้องกับคำ เพื่อบ่งบอกว่าคำนี้สิ้นสุดที่ตัวอักษรใด

2.3 ขั้นตอนการแปลงโครงสร้างข้อมูลเชิงวัตถุให้เป็นสายอักขระ (Java Serializable Library)

Serialization คือ กระบวนการในการแปลงวัตถุให้เป็นสายอักขระ (ไบต์สตรีม) เพื่อให้สามารถเก็บวัตถุในรูปของไฟล์ได้ ในบทความนี้จะทำการแปลงโครงสร้างข้อมูลแบบทรัยที่มีคลังคำอยู่ภายในโครงสร้างให้เป็นสายอักขระ (Serialization) วิธีนี้จะช่วยให้ไม่ต้องเพิ่มคลังคำเข้าไปในโครงสร้างทรัยทุกครั้งที่มีการตัดคำภาษาไทย เพียงแค่เรียกใช้งานสายอักขระให้กลับเป็นโครงสร้างทรัย (De-serialization) เมื่อต้องการตัดคำภาษาไทยเท่านั้น เพื่อความสะดวกและความรวดเร็วในการนำไปใช้ในการตัดคำภาษาไทย ซึ่งบทความนี้ได้นำการแปลงโครงสร้างข้อมูลเชิงวัตถุให้เป็นสายอักขระบนภาษาจาวาของแต่ละไลบรารี มาทั้งหมด 8 ไลบรารี [11] ได้แก่

1. FST จะเน้นเรื่องของ ความเร็ว ขนาดและความเข้ากันได้ของข้อมูลในการทำ serialization
2. GSON จะทำการแปลงข้อมูล Java objects เป็น JSON และทำการแปลง JSON เป็น Java objects
3. Protostuff เป็นการรองรับการทำงานแบบ forward-backward
4. Protobuf พัฒนาและใช้งานโดยกูเกิล รองรับภาษา Java, C++ และ Python มีความยืดหยุ่นและมีประสิทธิภาพ
5. Protostuff-json ทำการแปลงข้อมูล Java objects เป็น JSON และทำการแปลง JSON เป็น Java object
6. Protostuff-xml ทำการแปลงข้อมูล Java objects เป็น xml และทำการแปลง xml เป็น Java objects

7. Kryo-manual จะมีความเร็วและมีประสิทธิภาพในการทำ object graph serialization framework สำหรับจาวาในการ serialization
8. Java serialization เป็นไลบรารีของภาษาจาวา การ serialize ทำโดยการ implement java.io.Serializable ในคลาสที่ต้องการทำ serialization

โดยทำการวัดประสิทธิภาพเวลาการทำงานทั้งหมด 8 ไลบรารี เพื่อคัดเลือกไลบรารีที่มีประสิทธิภาพเวลาการทำงานเหมาะสำหรับนำมาใช้งานร่วมกับโครงสร้างทรัย

3. วิธีการดำเนินงาน

เนื่องจากประโยคหรือบทความภาษาไทยจะอยู่ในรูปแบบการเขียนแต่ละคำติดกัน ไม่เว้นช่องว่างระหว่างคำ ซึ่งแตกต่างจากภาษาอังกฤษที่มีช่องว่างระหว่างคำอย่างชัดเจน ดังตารางที่ 1

ตารางที่ 1. รูปแบบประโยคตัวอย่างและรายการคำ

ประโยคภาษาอังกฤษ	I went to rainforest
ประโยคภาษาไทย	ฉันเดินป่าดงดิบ
รายการคำ	ฉัน, เดิน, ป่า, ป่าดงดิบ, ดง, ดงดิบ, ดิบ

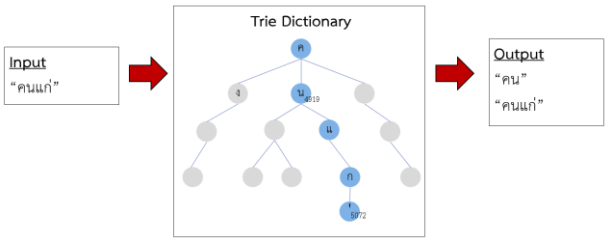
จากตารางที่ 1 รายการคำจะมีคำขึ้นต้นเหมือนกัน เช่นคำว่า “ป่า” และ “ป่าดงดิบ” ซึ่งเทคนิคการตัดคำที่เลือกใช้ คือ วิธีการตัดคำแบบยาวที่สุด (Longest matching) [12] และวิธีการย้อนรอยกลับ (Backtracking) [12] มาใช้ในการแก้ปัญหาคำกำกวมของคำในภาษาไทย

3.1 คลังคำ

พจนานุกรมเพื่อใช้ตัดคำภาษาไทยในบทความนี้นำคำมาจาก 2 แหล่งข้อมูลคือ 1. Lexitron [5] เป็นพจนานุกรมอิเล็กทรอนิกส์ที่ถูกพัฒนาโดยนักวิจัยจากศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติหรือเนคเทค (NECTEC) และ 2. คลังคำของโปรแกรม swath [6] เป็นพจนานุกรมที่นิยมนำมาใช้ในการพัฒนาโปรแกรม

3.2 การพัฒนาฟังก์ชัน findAllPrefix

ทรัยทั้ง 4 โครงสร้าง ขาดฟังก์ชันที่ใช้ในการค้นหาคำขึ้นต้นของประโยค จึงมีการพัฒนาฟังก์ชัน findAllPrefix เพิ่มลงในโครงสร้างทรัยซึ่งมีการทำงานดังตัวอย่างที่แสดงในรูปที่ 6



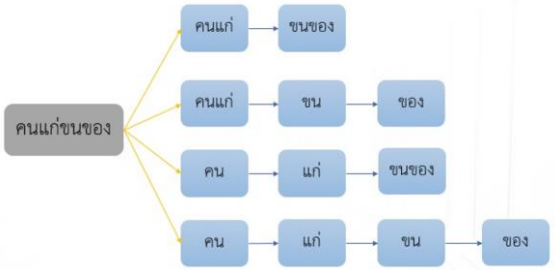
รูปที่ 6. ตัวอย่างกระบวนการค้นหา Prefix ของ “คนแก่ขนของ”

3.3 Safe segmentation/Unsafe segmentation

คำในภาษาไทยมักนิยมเขียนเป็นประโยคหรือบทความยาวๆ โดยไม่มีการเว้นช่องว่างระหว่างคำ ทำให้การตัดคำในภาษาไทยนั้นมีความซับซ้อนและเกิดปัญหา โดยการตัดคำในแต่ละครั้งจะไม่พบผลลัพธ์ของคำทุกคำในประโยคนั้นๆ และกรณีที่มีคำที่สะกดผิดในประโยคจะไม่มีการแสดงผลลัพธ์ในการตัดคำ บทความนี้ได้แบ่งรูปแบบการตัดคำภาษาไทยแบบพจนานุกรมออกเป็น 2 รูปแบบ คือ 1. แบบ Safe segmentation และ 2. แบบ Unsafe segmentation

3.3.1 การตัดคำแบบปลอดภัย (Safe segmentation)

เป็นวิธีการตัดคำที่ให้ผลลัพธ์เป็นทุกคำของการตัดคำ ซึ่งมีการทำงานแบบย้อนรอยและการตัดคำโดยให้ผลลัพธ์ของทุกความเป็นไปได้ในแต่ละประโยคหรือบทความนั้น แต่เมื่อพบคำที่สะกดผิด การตัดคำรูปแบบนี้จะไม่สามารถแสดงผลลัพธ์ใดๆ



รูปที่ 7. ตัวอย่างการตัดคำแบบปลอดภัย

การตัดคำแบบ Safe segmentation มีคุณสมบัติ ดังนี้

- ประโยคหรือบทความสามารถแทนด้วยคำภาษาไทยได้ทั้งหมด
- สามารถแสดงทุกรูปแบบของการจัดเรียงที่เป็นไปได้

ตัวอย่างผลลัพธ์ของการตัดคำแบบ Safe segmentation แสดงดังรูปที่ 7 และอัลกอริทึมของ Safe segmentation แสดงดังรูปที่ 8

การตัดคำแบบ Safe segmentation
1.รับประโยค
2.ตรวจสอบว่า ประโยคมีความยาวเท่ากับศูนย์หรือไม่ ถ้าเป็น จัดเก็บผลลัพธ์การตัดคำ
3.ถ้าไม่เป็น หาคำขึ้นต้นของประโยคในโครงสร้างทรัยโดยใช้ฟังก์ชัน findAllPrefix
    3.1 วนลูปคำขึ้นต้นทั้งหมด
    3.2 จัดเก็บคำขึ้นต้นประโยคลงในลิสต์
    3.3 นำคำขึ้นต้นไปตัดประโยคจะได้ประโยคใหม่และทำซ้ำข้อที่ 2-3
    3.4 ทำการย้อนกลับ ลบช่องล่าสุดของลิสต์ที่จัดเก็บคำเริ่มต้น ทำซ้ำข้อ 3

รูปที่ 8. ขั้นตอนวิธีการตัดคำแบบ Safe segmentation

ความสัมพันธ์เวียนเกิด (Recurrence relation) ของการตัดคำแบบ Safe segmentation แสดงดังสมการที่ 1

$$\text{Segment}_{\text{trie}}(\Sigma,\Omega) = \bigcup_{\rho \in \text{prefix}_{\text{trie},\Omega}} \text{Segment}_{\text{trie}}\,(\Sigma_{\text{append}\,(\rho)}, \Omega_{\text{substring}(\rho)}) \quad (1)$$

ให้เงื่อนไขเริ่มต้น (Initial condition) = $\text{Segment}_{\text{trie}}(\Sigma,\emptyset) = \Sigma$ และคำอธิบายสัญลักษณ์เป็นดังที่แสดงในตารางที่ 2

ตารางที่ 2. สัญลักษณ์สมการ

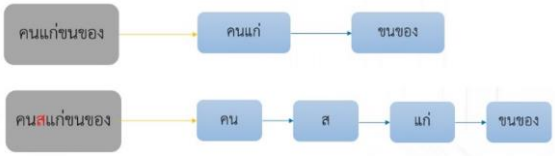
สัญลักษณ์	คำอธิบาย
Σ	ลำดับของคำ
Ω	ประโยคหรือบทความที่รับมา
prefix	คำขึ้นต้นของประโยคทั้งหมดในบทความนั้นๆ
$\emptyset$	เซตว่าง

3.3.2 การตัดคำแบบไม่ปลอดภัย (Unsafe segmentation)

การตัดคำแบบ Safe segmentation ไม่สามารถแก้ไขปัญหาการตัดคำในกรณีที่พบประโยคหรือบทความที่มีตัวสะกดผิด ซึ่งอาจเกิดจากการพิมพ์ผิดหรือภายในคลังคำไม่มีคำใหม่ จึงมีการเสนอวิธีการตัดคำแบบ Unsafe segmentation เพื่อแก้ไขปัญหานี้ การตัดคำแบบ Unsafe segmentation มีรูปแบบคำดังต่อไปนี้

- ประโยคหรือบทความไม่สามารถแทนด้วยคำไทยได้ทั้งหมด
- ข้ามอักขระทีละอักษรเมื่อพบตัวอักขระที่สะกดผิด

รูปที่ 9 แสดงตัวอย่างการตัดคำแบบ Unsafe segmentation



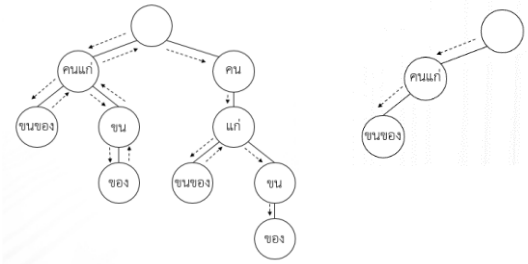
รูปที่ 9. ตัวอย่างการตัดคำแบบไม่ปลอดภัย

อัลกอริทึมของ Unsafe segmentation แสดงดังรูปที่ 10

การตัดคำแบบ Unsafe segmentation
1–2.การทำงานเหมือนรูปแบบ Safe segmentation
3.ถ้าไม่เป็น
    3.1 ตรวจสอบว่า ประโยคนี้มีคำขึ้นต้นประโยคหรือไม่ ถ้าไม่มี
        1) ทำการข้ามอักขระไป 1 อักขระ และจัดเก็บอักขระนั้น
        2) นำอักขระไปตัดประโยคจะได้ประโยคใหม่และทำซ้ำข้อที่ 2-3
    3.2 ถ้ามี หาคำขึ้นต้นของประโยคในโครงสร้างทรัยโดยใช้ฟังก์ชัน findAllPrefix
        1) วนลูปคำขึ้นต้นทั้งหมด
        2) จัดเก็บคำขึ้นต้นประโยคลงในลิสต์
        3) นำคำขึ้นต้นไปตัดประโยคจะได้ประโยคใหม่และทำซ้ำข้อที่ 2-3
        4) ทำการย้อนกลับจะหยุดการทำงาน

รูปที่ 10. ขั้นตอนวิธีการตัดคำแบบ Unsafe segmentation

การตัดคำภาษาไทยทั้งสองรูปแบบถูกนำมาใช้ในการพัฒนาโปรแกรมการตัดคำภาษาไทยที่มีการทำงานแบบเวียนเกิด โดยใช้เทคนิคการค้นหาแนวลึก (Depth first search) เข้ามาช่วยในการตัดคำภาษาไทยแบบพจนานุกรม รูปที่ 11 แสดงตัวอย่างการเปรียบเทียบการทำงานแบบการค้นหาแนวลึกของประโยค “คนแก่ขนของ”



การตัดคำแบบ Safe segmentation การตัดคำแบบ Unsafe segmentation

รูปที่ 11. เปรียบเทียบรูปแบบการตัดคำแบบ Safe segmentation กับ Unsafe segmentation

3.4 โมดูลการตัดคำ (Analyzer)

โมดูลการตัดคำเป็นคลาสบนจาวาที่ถูกสร้างขึ้นเพื่อใช้ในการตัดคำภาษาไทยแบบพจนานุกรมโดยเก็บข้อมูลในรูปแบบสาย

อักขระของโครงสร้างทรัย มีทั้งหมด 2 โมดูล ได้แก่ โมดูล SafeAnalyzer และโมดูล UnsafeAnalyzer ดังที่แสดงในตารางที่ 3

ตารางที่ 3. โมดูลที่ใช้ในการตัดคำ

Analyzer	โครงสร้างทรัย	รูปแบบการตัดคำภาษาไทย
SafeAnalyzer	TrieA	Safe segmentation
UnsafeAnalyzer	TrieA	Unsafe segmentation

4. ผลการดำเนินงาน

การทดสอบประสิทธิภาพของเวลาการทำงานใช้ประโยคทดสอบสองรูปแบบ คือ ประโยคหรือบทความที่สามารถแทนด้วยคำภาษาไทยได้ทั้งหมดและประโยคหรือบทความที่ไม่สามารถแทนด้วยคำภาษาไทยได้ทั้งหมด เพื่อให้เห็นความแตกต่างระหว่างการตัดคำ โดยใช้คำที่มีอยู่ในคลังคำ ดังนี้

4.1 ประโยคที่แทนด้วยคำภาษาไทยได้ทั้งหมด (Clean-sentence)

การสร้างรูปแบบประโยค จะสุ่มคำจากคลังคำมาเรียงต่อกันเป็นประโยคโดยไม่สนใจความหมายของประโยคนั้นๆ ดังประโยคตัวอย่างในรูปที่ 12

บัตรห้องสมุดติดิงพิซซ่าฮัทซัลเฟตเพ่งดูร้องรำทำเพลงบ่าวสาวถ่อมตัวเทศบาลเมืองความชื่นชม
หวันยิหวาผู้จ้างสำเนาสับสนปฏิกิริยาเคมีกุลาตีไม้อเจลกอินทรชิตมหาชัย

รูปที่ 12. ตัวอย่างประโยค Clean-sentence

4.2 ประโยคที่ไม่สามารถแทนด้วยคำภาษาไทยได้ทั้งหมด (Dirty-sentence)

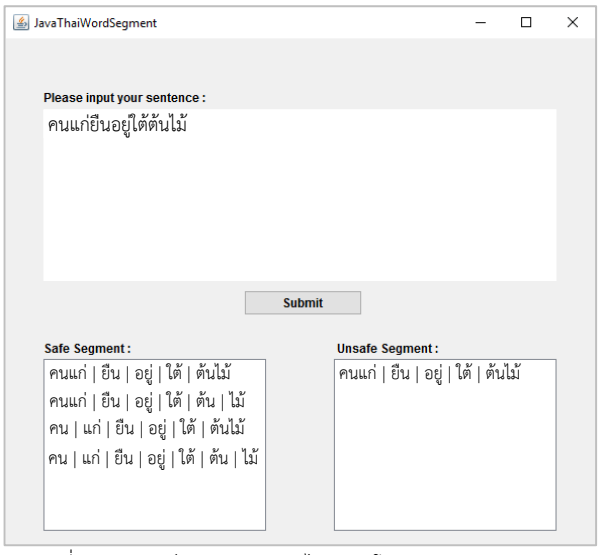
การสร้างรูปแบบประโยค จะสุ่มตัวอักษรภาษาอังกฤษแทรกไปในประโยคที่ทำการสุ่มมาเรียงต่อกัน โดยการสุ่มตัวอักษรภาษาอังกฤษจะช่วยให้ไม่เกิดคำที่ซ้ำซ้อนในภาษาไทย ดังตัวอย่างที่แสดงในรูปที่ 13

บัตรห้องสมุHดติดิงพิซซ่าฮัทซัลเฟตเพ่งดูร้องรำทำเพลงบ่าวสาวถ่อมตัวเทศบาลเมืองความชื่นชม
ตั้งใจฟังเหล่านี้กราวรูดซวนความมักง่ายแกนนำหEลับตาวอลล์ครหาหัก

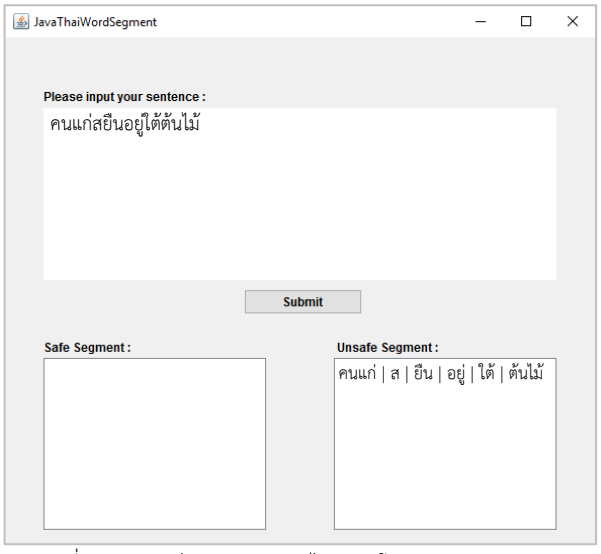
รูปที่ 13. ตัวอย่างประโยค Dirty-sentence

4.3 ผลลัพธ์การตัดคำ

การทดสอบประสิทธิภาพของโมดูลการตัดคำทั้งแบบ Safe segmentation และ Unsafe segmentation ใช้ประโยคทั้ง Clean-sentence และ Dirty-sentence ในการทดสอบ รูปที่ 14 และรูปที่ 15 แสดงตัวอย่างการทดสอบการตัดคำของประโยค “คนแก่ยืนอยู่ใต้ต้นไม้” แบบ Clean-sentence และแบบ Dirty-sentence ของทั้งสองโมดูล ซึ่งผลการทดสอบแสดงให้เห็นว่า Safe segmentation แสดงทุกความเป็นไปได้ของการตัดคำทั้งหมดแตกต่างจาก Unsafe segmentation ที่แสดงเพียงผลลัพธ์เดียว โดยเลือกคำที่ยาวที่สุดของแต่ละคำและถ้าพบอักขระที่สะกดผิดในประโยคจะทำการข้ามอักขระนั้น



รูปที่ 14. ผลลัพธ์การตัดคำภาษาไทยประโยค Clean-sentence



รูปที่ 15. ผลลัพธ์การตัดคำภาษาไทยประโยค Dirty-sentence

4.4 การวัดประสิทธิภาพ

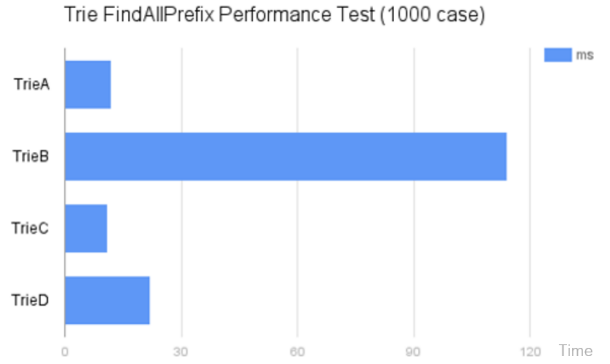
4.4.1 Platform

การวัดประสิทธิภาพการทำงาน ทำการวัดประสิทธิภาพบนเครื่องโน๊ตบุ๊คยี่ห้อ Dell รุ่น Vostro-3450

- การประมวลผล : Intel® Core™ i3-2310M CPU @ 2.10GHz × 4
- หน่วยความจำ : DDR3 4GB 1333 MHz
- ระบบปฏิบัติการ : Ubuntu 14.04 LTS 64-bit
- ดิสก์ : 312.8 GB

4.4.2 การวัดประสิทธิภาพของโครงสร้างทรัย

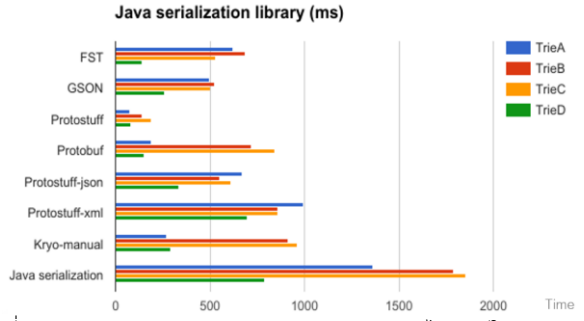
การวัดประสิทธิภาพของโครงสร้างทรัยทั้ง 4 โครงสร้าง ทำโดยทำการสุ่มคำที่มีอยู่ในคลังคำมาเรียงต่อกันเป็นประโยคที่มีความยาวประมาณ 150 ตัวอักษร ซึ่งมีทั้งหมด 1000 ประโยค โดยนำฟังก์ชัน findAllPrefix ของแต่ละโครงสร้างทรัยมาใช้ในการทดสอบ ผลการวัดประสิทธิภาพเวลาการทำงาน แสดงดังกราฟในรูปที่ 16



รูปที่ 16. กราฟแสดงประสิทธิภาพเวลาการทำงานของโครงสร้างทรัย ที่พัฒนาฟังก์ชัน findAllPrefix

4.4.3 การวัดประสิทธิภาพการแปลงสายอักขระให้เป็นโครงสร้างข้อมูลเชิงวัตถุ

การทดสอบประสิทธิภาพเวลาการทำงานของการแปลงสายอักขระให้เป็นโครงสร้างข้อมูลเชิงวัตถุ ใช้ในการเลือกไลบรารีที่ทำงานร่วมกับโครงสร้างทรัยที่มีคลังคำอยู่ภายในโครงสร้าง แล้วให้ประสิทธิภาพเวลาการทำงานที่รวดเร็วที่สุดเมื่อมีการเรียกใช้โปรแกรมตัดคำภาษาไทย ผลการวัดประสิทธิภาพแสดงดังกราฟในรูปที่ 17



รูปที่ 17. กราฟแสดงประสิทธิภาพเวลาการทำงานของไลบรารีในการแปลงสายอักขระให้เป็นโครงสร้างข้อมูลเชิงวัตถุ

4.4.4 การวัดประสิทธิภาพของโมดูลการตัดคำ

การวัดประสิทธิภาพการทำงานของโมดูลการตัดคำเป็นการวัดประสิทธิภาพของความเร็วในการทำงานของโมดูลการตัดคำ ซึ่งมี 2 รูปแบบ คือแบบ Safe segmentation และแบบ Unsafe segmentation ดังนั้น Analyzer ที่ใช้ในการวัดประสิทธิภาพจึงมีทั้งหมด 2 Analyzer ได้แก่ SafeAnalyzer และ Unsafe-Analyzer

โดยนำ Clean-sentence และ Dirty-sentence มาใช้ในการวัดความเร็วในการทำงานของ SafeAnalyzer และ UnsafeAnalyzer ตารางที่ 4 แสดงเวลาที่แต่ละ Analyzer ใช้ในการตัดคำของประโยค Clean และ Dirty-sentence โดยวัดจากการทดสอบ Clean และ Dirty-sentence ประเภทละ 1000 ประโยค

ตารางที่ 4. ประสิทธิภาพการทำงานของ SafeAnalyzer และ Unsafe-Analyzer (หน่วยเป็นมิลลิวินาที)

	SafeAnalyzer	UnsafeAnalyzer
Clean-sentence	15321 ms	21 ms
Dirty-sentence	345 ms	18 ms

ผลลัพธ์ในตารางที่ 4 แสดงให้เห็นว่าการตัดคำภาษาไทยของ Clean-sentence โดยใช้ SafeAnalyzer ใช้เวลาในการทำงานมากกว่าการใช้ UnsafeAnalyzer 700 เท่า เนื่องจากการตัดคำแบบ Safe segmentation ต้องแสดงคำทุกคำที่เป็นไปได้ในประโยคทั้งหมดจึงใช้เวลานานในการทำงาน ในขณะที่การตัดคำแบบ Unsafe segmentation จะแสดงผลลัพธ์เพียงหนึ่งผลลัพธ์เท่านั้นจึงใช้เวลาการตัดคำเร็วกว่าการตัดคำแบบ Safe segmentation มากและการตัดคำภาษาไทยโดยใช้

SafeAnalyzer ให้กับประโยคแบบ Dirty-sentence จะไม่มีผลลัพธ์เนื่องจากการตัดคำแบบ Safe segmentation ไม่สามารถตัดคำในประโยคที่มีการสะกดผิดได้

5. สรุป

บทความนี้นำเสนอวิธีการตัดคำภาษาไทยเพื่อใช้เป็นดัชนีฐานข้อมูล โดยใช้โครงสร้างทรัยในการสนับสนุนการตัดคำ โดยมีการพัฒนาฟังก์ชันในการหาคำขึ้นต้น findAllPrefix เพิ่มลงไปในโครงสร้างทรัยและมีการทดสอบหา 1. โครงสร้างทรัยที่ใช้เวลาเร็วที่สุดในการค้นหาข้อมูลร่วมกับ findAllPrefix และ 2. ไลบรารีการแปลงข้อมูลเชิงวัตถุให้เป็นสายอักขระที่ทำงานร่วมกับโครงสร้างที่เร็วที่สุด ซึ่งผลการทดสอบสรุปได้ว่าโครงสร้างข้อมูลแบบ TrieA ที่มีโครงสร้างแบบลิ้งค์ลิสต์และไลบรารี Protostuff ให้ผลลัพธ์ที่ดีที่สุด จึงนำ TrieA และ Protostuff มาใช้ในการสร้างโมดูลการตัดคำ ซึ่งในส่วนของอัลกอริทึมการตัดคำได้นำเสนออัลกอรึทึมการตัดคำจำนวน 2 อัลกอรึทึม คือ อัลกอรึทึม Safe segmentation ซึ่งช่วยในการแก้ปัญหาการตัดคำของประโยคที่มีคำกำกวมซึ่งให้ผลลัพธ์เป็นคำทุกคำในประโยคที่มีในพจนานุกรมและอัลกอรึทึม Unsafe segmentation เพื่อใช้ในการตัดคำของประโยคที่มีคำที่สะกดผิดหรือมีคำที่ไม่พบในพจนานุกรม ผลการทดสอบแสดงให้เห็นว่าอัลกอรึทึม Safe segmentation แม้ว่าจะใช้เวลาการทำงานที่สูงกว่าแต่ให้ผลลัพธ์เป็นคำทุกคำในประโยคที่มีในพจนานุกรม ในขณะที่อัลกอริทึม Unsafe segmentation จะมีการทำงานที่รวดเร็วแต่จะให้ผลลัพธ์เพียงคำตอบเดียว ในอนาคตจะมีการเพิ่มประสิทธิภาพเวลาการทำงานของ Safe segmentation ให้มีการทำงานที่รวดเร็วกว่าเดิม และมีการปรับปรุง Unsafe segmentation ให้สามารถแก้ไขคำที่สะกดผิดได้

เอกสารอ้างอิง

[1] S. Meknavin, P. Charoenpornsawat and B. Kijsirikul, “Feature-based Thai Word Segmentation”, Proceedings of the Natural Language Processing Pacific Rim Symposium 1997, December 1997, pp.41-46.

[2] T. Potipiti, V. Sornlertlamvanich, and T. Chaloenporn, “ Towards Building a Corpus- based Dictionary for Nonword- boundary Languages” , Proceedings of the Second International Conference on Language Resources and Evaluation (LREC2000), May 2000, pp. 82-86.

[3] V. Sornlertlamvanich, et. al Dictionary- less search engine for the collaborative database. In Proceedings of the Third International Symposium on Communications and Information Technologies, volume 1, pages 177–182, September.

[4] T. Theeramunkong and S. Usanavasin, “ Non-Dictionary Based Word Segmentation Using Decision Tree” , Proceedings of Human Language Technology (HLT 2001), March 2001.

[5] NECTEC, 2009, LEXiTRON Dictionary, [Online]. Available: http://lexitron.nectec.or.th/2009_1/.

[6] P. Charoenpornsawat, 2004, Software: SWATH – Thai Word Segmentation, [Online] . Available: http://www.cs.cmu.edu/~paisarn/software.html.

[7] M. Bhojasia, 2016, Java Program to Implement Trie, [Online]. Available: http://www.sanfoundry.com/java-program-implement-trie/.

[8] CODE REVIEW, 2014, Trie (tree form) in Java, [Online]. Available: http://codereview.stackexchange .com/questions/41630/trie-tree-form-in-java.

[9] R. C. Enaganti, 2013, Longest prefix matching – A Trie based solution in Java, [Online] . Available: http://www.geeksforgeeks.org/longest-prefix-matching-a-trie-based-solution-in-java/.

[10] R. Sedgewick and K. Wayne, Algorithms FOURTH EDITION, Massachusetts: Courier, 2011.

[11] Pascal S. de Kloe, 2016, jvm-serializers, [Online]. Available: https: / / github. com/ eishay/ jvm-serializers/wiki.