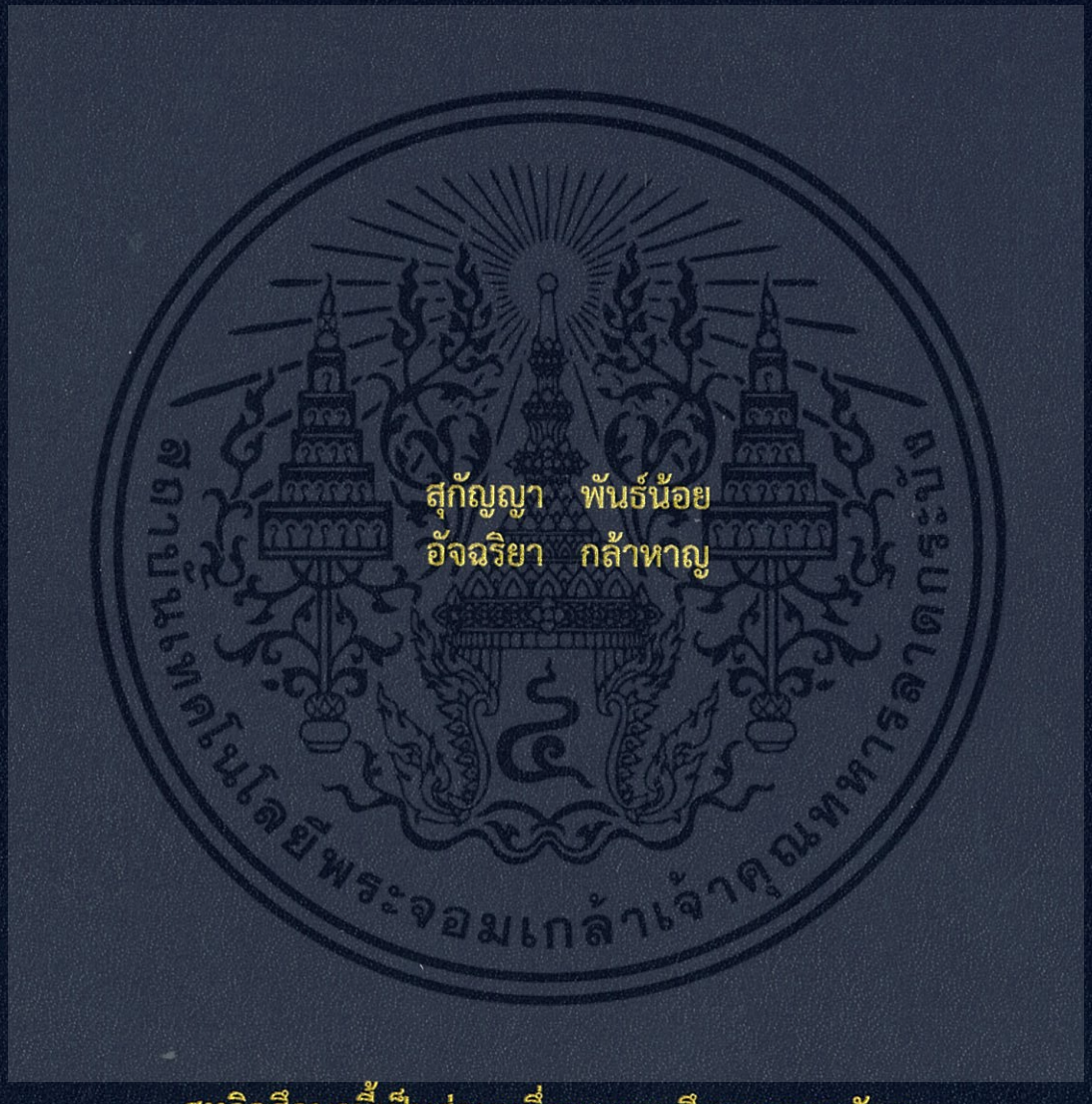
การตัดคำโดยใช้พจนานุกรมสำหรับทำดัชนีฐานข้อมูลเอกสารภาษาไทย

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



สุกัญญา พันธ์น้อย
อัจฉริยา กล้าหาญ

สหกิจศึกษานี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร
ปริญญาวิทยาศาสตรบัณฑิต (วิทยาการคอมพิวเตอร์)
ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2559

การตัดคำโดยใช้พจนานุกรมสำหรับทำดัชนีฐานข้อมูลเอกสารภาษาไทย

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

สุกัญญา พันธ์น้อย
อัจฉริยา กล้าหาญ

สหกิจศึกษานี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร
ปริญญาวิทยาศาสตรบัณฑิต(วิทยาการคอมพิวเตอร์)
ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2559

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

SUKUNYA PANNOI

ACHARIYA KLAHAN

COOPERATIVE EDUCATION SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR

หัวข้อสหกิจศึกษา	การตัดคำโดยใช้พจนานุกรมสำหรับทำดัชนีฐานข้อมูลเอกสารภาษาไทย Thai Word Segmentation for Data Indexing in Dictionary-based Search Engine
ชื่อนักศึกษา	นางสาวสุกัญญา พันธ์น้อย รหัสนักศึกษา 56050406 นางสาวอัจฉริยา กล้าหาญ รหัสนักศึกษา 56050425
ปริญญา	วิทยาศาสตรบัณฑิต (วิทยาการคอมพิวเตอร์)
ภาควิชา	วิทยาการคอมพิวเตอร์
ปีการศึกษา	2559
อาจารย์ที่ปรึกษา	ดร.รุ่งรัตน์ เวียงศรีพนาวัลย์

คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง (สจล.) อนุมัติให้สหกิจศึกษานี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญา วิทยาศาสตรบัณฑิต (วิทยาการคอมพิวเตอร์) ประจำปีการศึกษา 2559

คณะกรรมการสอบ	ลายมือชื่อ
ผศ.ดร.อนันตพร หรรษคุณาฒัย ประธานกรรมการ	อนันตพร หรรษคุณาฒัย
ดร.รุ่งรัตน์ เวียงศรีพนาวัลย์ กรรมการและอาจารย์ที่ปรึกษา	R Wi

หัวข้อสหกิจศึกษา การตัดคำโดยใช้พจนานุกรมสำหรับทำดัชนีฐานข้อมูลเอกสารภาษาไทย
ชื่อนักศึกษา นางสาวสุกัญญา พันธ์น้อย รหัสนักศึกษา 56050406
นางสาวอัจฉริยา กล้าหาญ รหัสนักศึกษา 56050425
ปริญญา วิทยาศาสตรบัณฑิต (วิทยาการคอมพิวเตอร์)
ภาควิชา วิทยาการคอมพิวเตอร์
คณะ วิทยาศาสตร์
มหาวิทยาลัย สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง (สจล.)
ปีการศึกษา 2559
อาจารย์ที่ปรึกษา ดร.รุ่งรัตน์ เวียงศรีพนาวัลย์

บทคัดย่อ

โครงงานสหกิจศึกษานี้เป็นโครงการของบริษัท ทิ้งค์เน็ต จำกัด มีวัตถุประสงค์เพื่อเสนอวิธีแก้ปัญหาคำกำกวมในการตัดคำภาษาไทยเพื่อใช้ในกระบวนการการตัดคำของการทำดัชนีแบบอินเวิร์ทภายในระบบเสิร์ชเอนจิ้น การทำงานประกอบด้วย 2 ส่วนหลัก ส่วนแรกคือการทดสอบหาโครงสร้างข้อมูลที่เหมาะสมในการสร้างโครงสร้างทรัยเพื่อใช้ในการสืบค้นคำในพจนานุกรม และ การทดสอบหาไลบรารีที่มีประสิทธิภาพที่ดีที่สุดในการแปลงข้อมูลเชิงวัตถุของโครงสร้างข้อมูลทรัยให้อยู่ในรูปสายอักขระเพื่อลดเวลาในการสร้างโครงสร้างทรัย ซึ่งผลการทดสอบพบว่า ลิ้งค์ลิสต์และโปรโตสตัฟฟ์เป็นโครงสร้างข้อมูลและไลบรารีที่มีประสิทธิภาพที่ดีที่สุด และส่วนที่สองคือการพัฒนาฟังก์ชัน findAllPrefix เพื่อใช้ในการสืบค้นคำขึ้นต้นในประโยคและพัฒนาขั้นตอนวิธีในการตัดคำ ได้แก่ ขั้นตอนวิธีการตัดคำแบบปลอดภัยซึ่งมีการนำเทคนิคการขยายและจำกัดเขตมาใช้ในการเพิ่มประสิทธิภาพเหมาะสำหรับข้อมูลที่ไม่มีการสะกดผิด และ ขั้นตอนวิธีการตัดคำแบบไม่ปลอดภัยเพื่อใช้ในการตัดคำในประโยคที่มีการสะกดผิดหรือไม่มีคำในพจนานุกรม โมดูลทั้งหมดพัฒนาด้วยภาษาจาวา

คำสำคัญ : การขยายและจำกัดเขต การตัดคำภาษาไทยโดยใช้พจนานุกรม การทำดัชนีแบบอินเวิร์ท การแปลงวัตถุให้เป็นสายอักขระ โครงสร้างทรัย

Title	THAI WORD SEGMENTATION FOR DATA INDEXING IN DICTIONARY-BASED SEARCH ENGINE
Students	Miss Sukunya Pannoi Student ID 56050406
	Miss Achariya Klahan Student ID 56050425
Degree	Bachelor of Science (Computer Science)
Department	Computer Science
Faculty	Science
University	King Mongkut's Institute of Technology Ladkrabang (KMITL)
Academic Year	2559
Advisor	Dr.Rungrat Wiangsripanawan

Abstract

This cooperation education project was done at Thinknet Company Limited. The project's objective is to propose a solution for word segmentation ambiguity using dictionary in Thai language of invert indexing within the search engine system. The work consists of two main parts. The first part is to find the most appropriate trie data structure to build the dictionary and the most effective library for converting object-oriented data structure into strings to reduce the time to build the structure. The result shows that link list and protostuff are the best data structure and library. The second part is to develop the findAllPrefix function to search the beginning word and to develop two word segmentation algorithms: the Safe segment algorithm using the branch and bound technique for sentences without misspelling and the Unsafe segment algorithm for sentences with misspelled words or words that cannot be found in the dictionary. All modules are developed in java.

Keyword: Branch and bound, Dictionary based Thai-word segmentation, content analyzer, Serialization, Trie structure

กิตติกรรมประกาศ

โครงการสหกิจศึกษานี้สำเร็จลุล่วงไปได้ด้วยดี เนื่องด้วยได้รับความช่วยเหลือและความกรุณาอย่างดียิ่งจาก ผศ.กฤษฎา บุศรา ดร.รุ่งรัตน์ เวียงศรีพนาวัลย์ และผศ.ดร.อนันตพร หรรษคุณาฒัยที่ได้ให้คำปรึกษาอย่างใกล้ชิด และเสนอแนะแนวทางแก้ปัญหา รวมทั้งตรวจสอบให้โครงการสหกิจศึกษานี้มีความสมบูรณ์เพิ่มขึ้น คณะผู้จัดจึงใคร่ขอขอบคุณท่านเป็นอย่างสูงไว้ ณ โอกาสนี้

ขอขอบคุณอาจารย์สาขาวิชาวิทยาการคอมพิวเตอร์ทุกท่านที่ได้ให้วิชาความรู้ และให้คำปรึกษาทั้งในภาคทฤษฎีและภาคปฏิบัติมาตลอด จนกระทั่งโครงการสหกิจศึกษานี้สัมฤทธิ์ผลได้ด้วยดีทุกประการ

ขอขอบคุณ คุณภูษิต ศรีดุลยกุลย์ ประธานบริษัท ทิ้งค์เน็ต จำกัด ที่เปิดโอกาสให้คณะผู้จัดทำได้มีส่วนเข้าไปเรียนรู้การทำงานจริง ซึ่งทำให้ผู้จัดทำได้รับประสบการณ์การทำงานที่เป็นประโยชน์อย่างยิ่ง คณะผู้จัดทำจึงใคร่ขอขอบคุณมา ณ โอกาสนี้

ขอขอบคุณ คุณกานต์ สกุลศักดิ์ คุณประพีร์พัฒน์ เอื้อวิจิตรพจนา คุณสรพล ชมไพศาล คุณณภัทร สุริยะพันธ์ คุณวสุ วินิจมนตรี และผู้มีส่วนเกี่ยวข้องในการพัฒนาการตัดคำภาษาไทยแบบคีย์เวิร์ดบนระบบการค้นหาข้อมูลทุกท่านที่ได้ให้คำปรึกษา รวมถึงให้แนวทางในการแก้ปัญหาและคอยดูแลอย่างใกล้ชิดในระหว่างระยะเวลาโครงการสหกิจศึกษา คณะผู้จัดทำจึงใคร่ขอขอบคุณมา ณ โอกาสนี้ด้วย

ท้ายที่สุดนี้ คณะผู้จัดทำขอกราบขอบพระคุณบิดา มารดาที่ได้ให้การสนับสนุนด้าน ทุนการศึกษา ให้คำปรึกษาและคอยเป็นกำลังใจที่สำคัญ ผู้จัดทำจึงใคร่ขอขอบพระคุณทุกท่านเป็นอย่างสูงไว้ ณ ที่นี้

สุกัญญา พันธ์น้อย
อัจฉริยา กล้าหาญ

สารบัญ

หน้า

บทคัดย่อภาษาไทย ..... ก
บทคัดย่อภาษาอังกฤษ..... ข
กิตติกรรมประกาศ..... ค
สารบัญ..... ง
สารบัญตาราง..... ช
สารบัญรูป ..... ซ
บทที่ 1 บทนำ..... 1
1.1 ความเป็นมาและความสำคัญ..... 1
1.2 วัตถุประสงค์ของงานวิจัย 1
1.3 ขอบเขตของงานวิจัย..... 2
1.4 ประโยชน์ที่ได้รับ..... 2
1.5 อุปกรณ์ที่ใช้ในการดำเนินงาน..... 3
1.6 ระยะเวลาและแผนการดำเนินงาน..... 3
1.7 นิยามศัพท์ของคำสำคัญ..... 4
บทที่ 2 ทฤษฎีและเทคโนโลยีที่เกี่ยวข้อง..... 5
2.1 ทฤษฎีและองค์ความรู้..... 5
2.1.1 การทำดัชนีแบบอินเวิร์ท (inverted index)..... 5
2.1.2 ทรัย (Trie)..... 7
2.1.3 Serialization..... 11
2.1.4 ความสัมพันธ์เวียนเกิด (Recurrence Relation)..... 18
2.1.5 การค้นตามแนวกว้าง (Breadth first search)..... 19
2.1.6 การค้นตามแนวลึก (Depth first search)..... 23

2.1.7 การขยายและจำกัดเขต (Branch and bound)..... 27
2.1.8 คิว (Queue)..... 28
2.1.9 การวิเคราะห์อัลกอริทึม (Algorithm Analysis)..... 34
2.1.10 การทดสอบประสิทธิภาพ (Performance Test)..... 46

สารบัญ (ต่อ)

หน้า

2.2 เครื่องมือ..... 47
2.2.1 Eclipse..... 47
2.2.2 EGit..... 48
บทที่ 3 วิธีการดำเนินงาน..... 51
3.1 ปัญหาที่เกิดขึ้น..... 51
3.2 แนวทางการแก้ปัญหา..... 51
3.2.1 ศึกษาทฤษฎีและโครงสร้างข้อมูลที่เกี่ยวข้อง..... 51
3.2.2 ความสัมพันธ์ของปัญหา..... 52
3.2.3 พัฒนาฟังก์ชันที่ตอบสนองความต้องการ..... 54
3.3 ขั้นตอนการดำเนินงานในการแก้ปัญหา..... 54
1. ขั้นตอนการสร้างพจนานุกรม..... 54
3.3.1 คลังคำ..... 55
3.3.2 วิเคราะห์หลักการทำงานของโครงสร้างข้อมูลแบบทรัย..... 55
3.3.3 การแปลงโครงสร้างข้อมูลเชิงวัตถุให้อยู่ในรูปสายอักขระ..... 63
2. ขั้นตอนการตัดคำ..... 72
3.3.4 พัฒนาฟังก์ชัน findAllPrefix..... 73
3.3.5 การวิเคราะห์อัลกอริทึมฟังก์ชัน findAllPrefix..... 80

3.3.6 พัฒนาฟังก์ชันการตัดคำภาษาไทย 80
3.3.7 โมดูลการตัดคำ (Analyzer) 92
3.3.8 พัฒนาฟังก์ชันการตัดคำภาษาไทยโดยเทคนิคการขยายและจำกัดเขต (Branch and bround) 94
3.3.9 Demo 96
บทที่ 4 ผลการดำเนินงาน 98
4.1 การวิเคราะห์อัลกอริทึม 98
4.2 การวัดประสิทธิภาพการทำงานของโครงสร้างข้อมูลแบบ ..... ทรัย 99

สารบัญ (ต่อ)

หน้า

4.3 การวัดประสิทธิภาพการทำงานของการแปลงโครงสร้างข้อมูลเชิงวัตถุให้เป็นสายอักขระ 100
4.4 การวัดประสิทธิภาพการทำงานของโมดูลการตัดคำ 101
4.5 การวัดประสิทธิภาพการทำงานของการขยายและจำกัดเขต (Branch and Bound) 103
บทที่ 5 สรุปผลการดำเนินงานและข้อเสนอแนะ 106
5.1 การดำเนินงานจัดทำโครงงาน 106
5.1.1 วัตถุประสงค์ของโครงงาน 106
5.1.2 วัสดุ อุปกรณ์ เครื่องมือหรือโปรแกรมที่ใช้ในการพัฒนาโครงงาน 106
5.2 สรุปผลการดำเนินงาน 107
5.3 ข้อจำกัดของการพัฒนาและข้อเสนอแนะ 107
5.4 สรุปผลการปฏิบัติงาน 108
เอกสารอ้างอิง 109
ภาคผนวก 111

สารบัญตาราง

ตารางที่ **หน้า**

1.1 แผนการดำเนินงาน 3

1.2 ตารางแสดงคำศัพท์เฉพาะด้าน 4

3.1 สัญลักษณ์สมการ 53

4.1 แสดงประสิทธิเวลาการทำงานของโมดูลการตัดคำระหว่าง Safe segment และ Unsafe segment 102

4.2 แสดงประสิทธิเวลาการทำงานของโมดูลการตัดคำในกรณีที่ดีที่สุดและกรณีที่แย่ที่สุด 103

4.3 แสดงประสิทธิเวลาการทำงานของโมดูลการตัดคำรูปแบบ Safe segment ที่ใช้เทคนิคการตัดคำแต่ละแบบ 104

สารบัญรูป

รูปที่ **หน้า**

2.1 กระบวนการพื้นฐานของ Search Engine..... 5
2.2 แสดงส่วนประกอบหลักของการทำดัชนีแบบอินเวิร์ท (Inverted Indexing)..... 7
2.3 โครงสร้างข้อมูลแบบทรัย..... 8
2.4 ตัวอย่างการค้นหาในทรัย..... 9
2.5 การเพิ่มข้อมูลในทรัย..... 11
2.6 Trie representation (R = 26)..... 12
2.7 implements Serializable..... 13
2.8 การทำ De-serialization..... 14
2.9 implements Serializable..... 15
2.10 การทำ Transient..... 16
2.11 implements Exterrializable..... 17
2.12 การทำ serialization..... 18
2.13 แสดงการแก้ไขคลาส A..... 18
2.14 การทำ de-serialization..... 18
2.15 ตัวอย่างต้นไม้ปริภูมิสถานะ..... 21
2.16 การค้นหาตามแนวกว้าง (Breadth first search)..... 22
2.17 รหัสเทียมค้นคำตอบของปัญหาผลรวมของเซตย่อยแบบตามแนวกว้าง..... 23
2.18 การค้นตามแนวกว้างของเพื่อหาเซตย่อยของ {25, 10, 9, 2} ที่มีผลรวมเป็น 36..... 24
2.19 ตัวอย่างต้นไม้ปริภูมิสถานะ..... 25
2.20 การค้นหาตามแนวลึก (Depth first search)..... 26
2.21 รหัสเทียมค้นคำตอบของปัญหาผลรวมของเซตย่อยแบบตามแนวลึก..... 27
2.22 ตัวอย่างการใช้ขอบเขตของผลเฉลยเพื่อเล็มต้นไม้..... 28
2.23 หลักการของคิว..... 30
2.24 แสดงการเพิ่มข้อมูลลงในคิวด้วยฟังก์ชัน Enqueue..... 31
2.25 แสดงการลบข้อมูลออกจากคิวด้วยฟังก์ชัน Dequeue..... 32
2.26 แสดงการดึงข้อมูลส่วนหัวคิวออกมาใช้งานด้วยฟังก์ชัน Queue front..... 33

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

2.27 แสดงการดึงข้อมูลส่วนท้ายคิวออกมาใช้งานด้วยฟังก์ชัน Queue rear..... 34
2.28 ตัวอย่างการทำงานของคิวด้วยฟังก์ชันต่างๆ..... 35
2.29 โครงสร้างข้อมูลแบบเชิงเส้น..... 36
2.30 โครงสร้างข้อมูลแบบไม่เป็นเชิงเส้น..... 37
2.31 ตัวอย่าง Pseudo Code 1..... 37
2.32 ตัวอย่าง Pseudo Code 2..... 37
2.33 ตัวอย่างการวิเคราะห์ Space Complexity..... 39
2.34 ตัวอย่างการทำงานแบบ recursive..... 40
2.35 การทำงานแบบ Linear Loops..... 41
2.36 การทำงานแบบ Logarithmic Loops..... 42
2.37 ตัวอย่างการคำนวณค่า..... 44
2.38 กราฟแสดงอัตราการเติบโต Big-O..... 45
2.39 กราฟแสดงอัตราการเติบโต Big-Omega..... 46
2.40 กราฟแสดงอัตราการเติบโต Big-Theta..... 47
2.41 แสดงการเติบโตของฟังก์ชันที่ใช้บ่อยๆในการค่าประมาณ Big-O..... 48
3.1 ตัวอย่างการตัดคำแบบ Safe segmentation..... 52
3.2 ตัวอย่างการตัดคำแบบ Unsafe segmentation 53
3.3 แผนภาพการทำงานการตัดคำภาษาไทยแบบคีย์เวิร์ดบนระบบการค้นหาข้อมูล..... 54
3.4 .....แผนภาพ
การสร้างพจนานุกรมสำหรับการตัดคำภาษาไทย..... 55
3.5 TrieNode ที่มีการทำงานแบบ .....ลิ้งค์ลิสต์
..... 54
3.6 ตัวอย่างโครงสร้างการทำงานของ TrieA..... 57
3.7 TrieNode ที่มีการทำงานแบบแฮ.....ชแม็พ
..... 59
3.8 ตัวอย่างโครงสร้างการทำงานของ TrieB และ TrieC 60

3.9 TrieNode ที่มีการทำงานแบบ Node tree 61
3.10 ตัวอย่างโครงสร้างการทำงานของ TrieD 62
3.11 การแปลงโครงสร้างข้อมูลเชิงวัตถุให้อยู่ในรูปสายอักขระของ GSON 64

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

3.12 การแปลงโครงสร้างข้อมูลเชิงวัตถุให้อยู่ในรูปสายอักขระของ Protobuf 65
3.13 การแปลงโครงสร้างข้อมูลเชิงวัตถุให้อยู่ในรูปสายอักขระของ Protostuff 66
3.14 การแปลงโครงสร้างข้อมูลเชิงวัตถุให้อยู่ในรูปสายอักขระของ Protostuff-json 67
3.15 การแปลงโครงสร้างข้อมูลเชิงวัตถุให้อยู่ในรูปสายอักขระของ Protostuff-xml 68
3.16 การแปลงโครงสร้างข้อมูลเชิงวัตถุให้อยู่ในรูปสายอักขระของ FST 69
3.17 การแปลงโครงสร้างข้อมูลเชิงวัตถุให้อยู่ในรูปสายอักขระของ Kryo 70
3.18 การแปลงโครงสร้างข้อมูลเชิงวัตถุให้อยู่ในรูปสายอักขระของ Java serialize 71
3.19 แผนภาพขั้นตอนการตัดคำภาษาไทย 72
3.20 แผนภาพการทำงานฟังก์ชัน findAllPrefix 73
3.21 ตัวอย่างกระบวนการค้นหา Prefix 74
3.22 ฟังก์ชัน findAllPrefix หลักการทำงานแบบลูป for ของ TrieD 75
3.23 ฟังก์ชัน findAllPrefix แบบ Optimization ของ TrieA 76
3.24 ฟังก์ชัน findAllPrefix แบบ Optimization ของ TrieB 77
3.25 ฟังก์ชัน findAllPrefix แบบ Optimization ของ TrieC 78
3.26 ฟังก์ชัน findAllPrefix แบบ Optimization ของ TrieD 79
3.27 การทำงานแบบค้นหาตามแนวลึกของประโยค 81
3.28 ฟังก์ชัน segment หลักการทำงานแบบค้นหาตามแนวลึก กรณี Safe segmentation 82
3.29 ฟังก์ชัน segment หลักการทำงานแบบค้นหาตามแนวลึก กรณี Unsafe segmentation. 83
3.30 การทำงานแบบค้นหาตามแนวกว้างของประโยค 84
3.31 ตัวอย่างกระบวนการทำงานการตัดคำภาษาไทยทำงานร่วมกับคิว 85
3.32 ฟังก์ชัน segment หลักการทำงานแบบค้นหาตามแนวลึก กรณี Safe segmentation 89
3.33 ฟังก์ชัน segment หลักการทำงานแบบค้นหาตามแนวลึก กรณี Unsafe segmentation 91
3.34 ตัวอย่างประโยค Clean-sentence 93

3.35 ตัวอย่างประโยค Dirty-sentence..... 93
3.36 การทำงานแบบการขยายและจำกัดเขตของประโยค..... 94

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

3.37 ฟังก์ชัน segment หลักการทำงานแบบค้นหาแนวลึกเพิ่มประสิทธิภาพของการตัดคำภาษาไทยด้วยเทคนิคการขยายและจำกัดเขต..... 95
3.38 รูปแบบหน้าต่างโปรแกรมการตัดคำแบบ Input..... 96
3.39 รูปแบบหน้าต่างโปรแกรมการตัดคำแบบ Output..... 97
4.1 กราฟแสดงประสิทธิภาพเวลาการทำงานของโครงสร้างข้อมูลแบบ.....ทรัย
.....99
4.2 กราฟแสดงประสิทธิภาพเวลาการทำงานของ Deserialization..... 101
ก.1 หน้าเว็บไซต์ Download Java JDK..... 113
ก.2 ขั้นตอนการติดตั้ง Java JDK (1)..... 114
ก.3 ขั้นตอนการติดตั้ง Java JDK (2)..... 114
ก.4 ขั้นตอนการติดตั้ง Java JDK (3)..... 115
ก.5 ขั้นตอนการติดตั้ง Java JDK (4)..... 115
ก.6 ขั้นตอนการติดตั้ง Java JDK (5)..... 116
ก.7 ขั้นตอนการติดตั้ง Java JDK (6)..... 116
ข.1 หน้าเว็บไซต์ Download Eclipse..... 118
ข.2 หน้าเว็บไซต์ Download Eclipse เลือก version..... 119
ข.3 ขั้นตอนการติดตั้ง Eclipse (1)..... 120
ข.4 ขั้นตอนการติดตั้ง Eclipse (2)..... 121
ข.5 ขั้นตอนการติดตั้ง Eclipse (3)..... 121
ข.6 ขั้นตอนการติดตั้ง Eclipse (4)..... 122
ค.1 การติดตั้ง Plugin Git บน Eclipse..... 124
ค.2 สร้าง repository บน GitHub..... 125
ค.3 Open Perspective บน GitHub..... 126
ค.4 ทำการ Clone git..... 126

ค.5 นำ source code โปรเจคเข้า Git..... 127
ค.6 การ commit code..... 128
ค.7 commit ไม่สำเร็จ..... 128

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

ค.8 การ push ลง Git..... 129
ค.9 การ Clone Git..... 130
ค.10 import project..... 131

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญ

ในปัจจุบันเป็นยุคแห่งเทคโนโลยีและสังคมออนไลน์ ข้อมูลต่างๆ ได้กระจายอยู่ทั่วไปบนระบบเครือข่ายอินเทอร์เน็ต ทั้งข้อมูลที่เป็นตัวอักษรและข้อมูลมัลติมีเดีย เช่น รูปภาพ วีดีโอ และเสียง ซึ่งมีแนวโน้มที่จะเพิ่มมากขึ้นอย่างรวดเร็ว อย่างไรก็ตามการมีข้อมูลจำนวนมากบนระบบเครือข่ายอินเทอร์เน็ต นำปัญหามาสู่ผู้ใช้ นั่นคือผู้ใช้ไม่สามารถที่จะค้นหาข้อมูลที่ต้องการได้อย่างถูกต้อง สะดวกและรวดเร็ว โดยในปัจจุบันโปรแกรมที่นิยมใช้ในการค้นหาข้อมูลบนระบบเครือข่ายอินเทอร์เน็ต คือ เสิร์ชเอนจิน (search engine) เว็บไซต์ที่ใช้การค้นหาแบบเสิร์ชเอนจิน เช่น Google Yahoo เป็นต้น

เสิร์ชเอนจิน คือ โปรแกรมที่ช่วยในการสืบค้นหาข้อมูล โดยเฉพาะข้อมูลบนอินเทอร์เน็ต โดยครอบคลุมทั้งข้อความ รูปภาพ ภาพเคลื่อนไหว เพลง ซอฟต์แวร์ และอื่นๆ ซึ่งแตกต่างกันไป โดยปกติแล้วเสิร์ชเอนจินส่วนใหญ่จะไม่ทำการค้นหาข้อมูลจากเอกสารโดยตรง แต่จะค้นหาบนข้อมูลที่เป็นตัวแทนเอกสารต่างๆ ในฐานข้อมูล เรียกข้อมูลตัวแทนนั้นว่า การทำดัชนีแบบอินเวิร์ท (inverted index) [3] ในการทำดัชนีแบบอินเวิร์ทมีกระบวนการทำงานโดยทำการตัดคำ (Tokenization) [1],[2] ในเอกสารออกเป็นคำๆ สำหรับกระบวนการนี้ในภาษาอังกฤษนั้นถือว่าเป็นขั้นตอนที่ง่าย เพราะคำในภาษาอังกฤษมีการเว้นวรรคและช่องว่างระหว่างคำอย่างชัดเจน แต่คำในภาษาไทยนั้นเป็นบทความที่ติดกันไม่มีช่องว่างระหว่างคำ ทำให้เกิดปัญหาในการตัดคำภาษาไทย

โปรแกรมการตัดคำภาษาไทยแบบคีย์เวิร์ดบนระบบการค้นหาข้อมูล เป็นโปรแกรมที่ใช้สำหรับตัดคำภาษาไทยเพื่อนำไปใช้งานในกระบวนการการตัดคำของการทำดัชนีแบบอินเวิร์ท โดยใช้โครงสร้างข้อมูลแบบทรัย (Trie Structure) มาเป็นโครงสร้างข้อมูลหลักที่ใช้ในการสืบค้นคำสำหรับการตัดคำภาษาไทย

1.2 วัตถุประสงค์ของงานวิจัย

1) เพื่อประยุกต์การตัดคำภาษาไทยในการสร้างดัชนีแบบอินเวิร์ทได้หลายรูปแบบและค้นหาคำภาษาไทยได้ตามความต้องการ
2) เพื่อพัฒนาโปรแกรมตัดคำภาษาไทยได้อย่างรวดเร็ว
3) เพื่อศึกษาและพัฒนาอัลกอริทึม (Algorithm) ที่ใช้ในการพัฒนาโปรแกรมตัดคำภาษาไทย

4) เพื่อศึกษา วัดประสิทธิภาพเวลาการประมวลผลและนำโครงสร้างข้อมูลแบบทรัยมาใช้ในการค้นหาคำภาษาไทยในพจนานุกรม
5) เพื่อศึกษาและวัดประสิทธิภาพเวลาการประมวลผลการแปลงโครงสร้างข้อมูลเชิงวัตถุให้เป็นสายอักขระบนภาษาจาวาของแต่ละไลบารี (Java Serializable Library) ที่มีประสิทธิภาพเหมาะแก่การนำมาใช้งาน

1.3 ขอบเขตของงานวิจัย

1) ตัดคำภาษาไทยโดยใช้การเทียบสายอักขระ
2) ตัดคำภาษาไทยจากประโยคหรือบทความที่สามารถแทนด้วยคำไทยได้ทั้งหมดให้แสดงทุกรูปแบบของการจัดเรียง
3) ตัดคำภาษาไทยจากประโยคหรือบทความที่ไม่สามารถแทนด้วยคำไทยได้ทั้งหมด
4) จัดเก็บคำในพจนานุกรมโดยใช้โครงสร้างข้อมูลแบบทรัย
5) ลบและเพิ่มคำที่มีอยู่ในพจนานุกรมได้
6) วัดประสิทธิภาพแง่เวลาการประมวลผลโครงสร้างข้อมูลสำเร็จรูปแบบทรัยในแต่ละโครงสร้าง
7) วัดประสิทธิภาพแง่เวลาการประมวลผลการแปลงโครงสร้างข้อมูลทรัยให้อยู่ในรูปสายอักขระของแต่ละชุดคำสั่ง
8) พัฒนาโปรแกรมการตัดคำแบบการเรียกซ้ำ (Recursion) โดยใช้เทคนิคการค้นหาตามแนวกว้าง (Breath First Search) และการค้นหาตามแนวลึก (Depth First Search)
9) เพิ่มประสิทธิภาพของโปรแกรมการตัดคำโดยใช้เทคนิคการขยายและจำกัดเขต (Branch and Bound)
10) พัฒนาฟังก์ชัน findAllPrefix
11) วัดประสิทธิภาพแง่เวลาการทำงานโปรแกรมตัดคำภาษาไทยของแต่ละเทคนิคที่นำมาใช้ในการพัฒนา
12) วิเคราะห์ประสิทธิภาพแง่เวลาการทำงานของอัลกอริทึมฟังก์ชัน findAllPrefix

1.4 ประโยชน์ที่ได้รับ

1) เพิ่มความรวดเร็วในการค้นหาคำภาษาไทย
2) สามารถตัดคำที่ต้องการจะค้นหาได้ตามความต้องการ
3) ได้เรียนรู้และเข้าใจโครงสร้างข้อมูลแบบทรัย

4) ได้เรียนรู้การแปลงโครงสร้างข้อมูลเชิงวัตถุให้เป็นสายอักขระบนภาษาจาวา
5) ได้เรียนรู้การพัฒนาโปรแกรมแบบการเรียกซ้ำโดยใช้เทคนิคการค้นหาตามแนว และการค้นหาตามแนวลึก
6) ได้เรียนรู้วิธีการเพิ่มประสิทธิภาพของโปรแกรมด้วยวิธีการขยายและจำกัดเขต

1.5 อุปกรณ์ที่ใช้ในการดำเนินงาน

ฮาร์ดแวร์ (Hardware)

Notebook ยี่ห้อ Dell รุ่น Vostro-3450 จำนวน 2 เครื่อง

- Processor : Intel® Core™ i3-2310M CPU @ 2.10GHz × 4
- Ram : DDR3 4GB 1333 MHz
- OS type : Ubuntu 14.04 LTS 64-bit
- Disk : 312.8 GB

ซอฟแวร์ (Software)

- Ubuntu 14.04.5 LTS
- Eclipse Java EE IDE for Web Developers
  Version: Mars.2 Release (4.5.2)
- EGit Version: 4.3.1.2016050

1.6 ระยะเวลาและแผนการดำเนินงาน

วันที่ 1 มิถุนายน พ.ศ. 2559 ถึง วันที่ 30 ธันวาคม พ.ศ. 2559
ประจำภาคเรียนที่ 1 ปีการศึกษา 2559
โดยมีรายละเอียดและเวลาที่ใช้ในการดำเนินการพัฒนาโปรแกรม ดังตาราง 1.1

ตารางที่ 1.1 แผนการดำเนินงาน

รายการ	2559						
	มิ.ย.	ก.ค.	ส.ค.	ก.ย.	ต.ค.	พ.ย.	ธ.ค.
1. ศึกษาเรียนรู้และทำความเข้าใจเกี่ยวกับทฤษฎีที่เกี่ยวข้อง	←	→					
2. ค้นคว้าและรวบรวมโครงสร้างข้อมูลแบบทรัยและการแปลง		↔					

โครงสร้างข้อมูลเชิงวัตถุให้เป็นสายอักขระบนภาษาจาวาของแต่ละไลบารี							
3. ทดสอบและวัดประสิทธิภาพเวลาการทำงานของโครงสร้างข้อมูลแบบทรัยและการแปลงโครงสร้างข้อมูลเชิงวัตถุให้เป็นสายอักขระบนภาษาจาวาของแต่ละไลบารี			←				
4. พัฒนาโปรแกรมตัดคำภาษาไทย				←	←		
5. เปรียบเทียบการนำโปรแกรมตัดคำภาษาไทยที่ใช้แต่ละเทคนิค						←	
6. สรุปผลและจัดทำเอกสารฉบับสมบูรณ์							←

1.7 นิยามศัพท์ของคำสำคัญ

ตารางที่ 1.2 ตารางแสดงคำศัพท์เฉพาะด้าน

ลำดับ	คำศัพท์	ความหมาย
1	Prefix	คำขึ้นต้นของประโยคทั้งหมดในบทความนั้นๆ
2	Safe segment	ประโยคหรือบทความที่สามารถแทนด้วยคำไทยได้ทั้งหมด
3	Unsafe segment	ประโยคหรือบทความที่ไม่สามารถแทนด้วยคำไทยได้ทั้งหมด
4	Anomaly	อักขระพิเศษ
5	Clean sentence	ประโยคในรูปแบบทั่วไป
6	Dirty sentence	ประโยคที่มีอักขระพิเศษอยู่ในประโยค

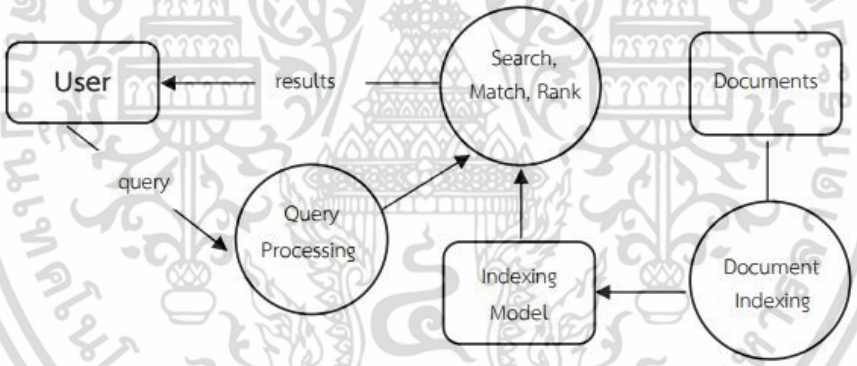
บทที่ 2
ทฤษฎีและเทคโนโลยีที่เกี่ยวข้อง

ในบทนี้จะกล่าวถึงรายละเอียดของทฤษฎีที่เกี่ยวข้องและเทคนิคต่างๆ ที่นำมาใช้ในการดำเนินงานพัฒนาโครงการการตัดคำภาษาไทยแบบคีย์เวิร์ดบนระบบการค้นหาข้อมูล ที่ได้รับมอบหมายจากการปฏิบัติสหกิจครั้งนี้ ซึ่งมีรายละเอียดดังต่อไปนี้

2.1 ทฤษฎีและองค์ความรู้

2.1.1 การทำดัชนีแบบอินเวิร์ท (inverted index)

การทำดัชนีจะทำเป็นอันดับแรกสุดของขั้นตอนการค้นคืนข้อมูลและมีความสำคัญอย่างยิ่งต่อประสิทธิภาพการค้นหาข้อมูลของระบบเพื่อใช้ใน Search Engine องค์ประกอบของ Search Engine สามารถแสดงในรูปแบบของ แผนภาพในรูปที่ 2.1



รูปที่ 2.1 กระบวนการพื้นฐานของ Search Engine

การทำดัชนีข้อมูล

กระบวนการทำดัชนีมีจุดประสงค์เพื่อหลีกเลี่ยงการค้นหาเอกสารทีละรายการตั้งแต่เอกสารอันดับที่หนึ่งจนถึงเอกสารอันดับสุดท้าย หรือเรียกว่า การค้นหาแบบเรียงลำดับ (Sequential Search) หรือการสแกนแบบเส้นตรง (Linear Scanning) ซึ่งเป็นวิธีการที่เสียเวลาและไม่มีประสิทธิภาพ ดังนั้นการทำดัชนีจึงช่วยลดเวลาในการค้นหาข้อมูลที่ต้องการ ในการทำดัชนีนี้องค์ประกอบหนึ่งที่มีอิทธิผลต่อการเลือกใช้หรือออกแบบอัลกอริทึมในการทำดัชนีคือ ฮาร์ดแวร์ของเครื่องคอมพิวเตอร์ที่จะใช้ในการทำดัชนี

วิธีการทำดัชนีที่รู้จักกันแพร่หลายคือ วิธีการทำดัชนีแบบอินเวิร์ท[4] ซึ่งมีขั้นตอนการทำงานดังต่อไปนี้

1.1 ทำการรวบรวมเอกสารที่ต้องการจะทำดัชนี

1.2 ทำการตัดคำ (Tokenizing) ในเอกสารออกเป็นคำๆ สำหรับภาษาอังกฤษนั้นขั้นตอนนี้เป็นขั้นตอนที่ง่าย แต่สำหรับภาษาอื่น เช่น ภาษาไทย ซึ่งเป็นภาษาที่ในประโยคไม่มีช่องว่างระหว่างคำ ขั้นตอนนี้ถือเป็นขั้นตอนที่ค่อนข้างยาก เช่น ตากลม ควรจะแบ่งคำออกเป็น ตา-กลม หรือ ตาก-ลม เป็นต้น ดังนั้น นักวิจัยไทยพยายามคิดค้นเทคนิคที่จะตัดคำให้ได้อย่างถูกต้องตามบริบท (Context) เพื่อที่จะสร้างดัชนีได้อย่างถูกต้อง ผลลัพธ์ของขั้นตอนนี้คือแต่ละเอกสารจะถูกเปลี่ยนเป็นชุดของคำต่างๆ สำหรับภาษาอังกฤษนั้นขั้นตอนนี้มีความท้าทายอีกประการหนึ่งคือ คำบางคำไม่ควรแยกออกจากกัน เช่น Great Britain เป็นชื่อประเทศ ดังนั้น คำนี้ควรถือเป็นคำเดียวและทำดัชนีเป็นคำเดียวกัน แนวทางแก้ปัญหานี้คือการใช้เทคนิคของการประมวลผลภาษาธรรมชาติ (Natural Language Processing) มาช่วยในการวิเคราะห์คำ

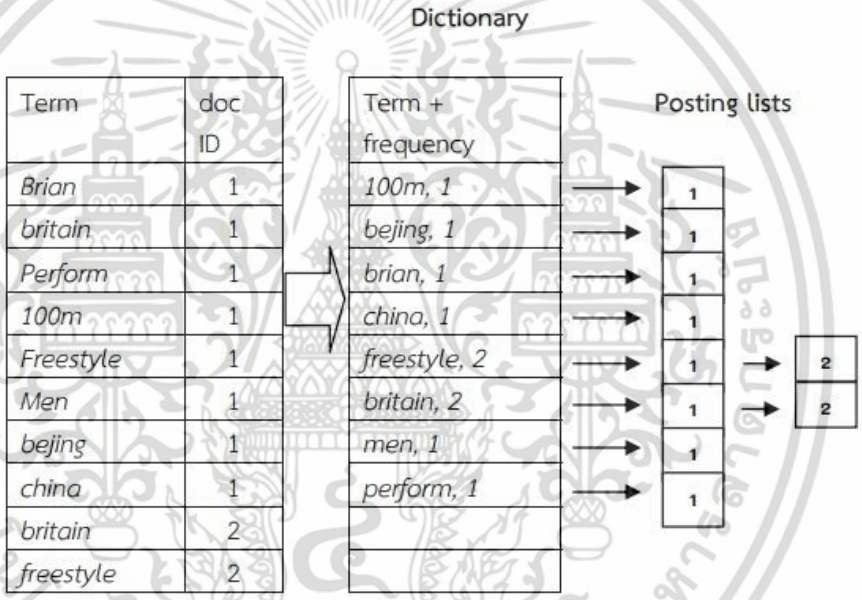
1.3 นอกจากนี้ยังมีการทำประมวลผลทางภาษา (Linguistic Processing) เพื่อเปลี่ยนรูปคำให้อยู่ในรูปแบบดั้งเดิม (Stemming) เช่น เปลี่ยนจากพหูพจน์ให้เป็นรูปแบบเอกพจน์ธรรมดา เปลี่ยนตัวพิมพ์ใหญ่เป็นตัวพิมพ์เล็ก เปลี่ยนคำที่ลงท้ายด้วย ing เป็นคำเดิม เช่น Swimming เป็น Swim เป็นต้น

1.4 ทำการตัดคำที่ไม่สำคัญทิ้งไป คำที่ไม่สำคัญ (Stop Words) คือ คำที่ไม่มีความหมายต่อเนื้อหาในเอกสารนั้นๆ และสามารถตัดทิ้งได้การตัดคำที่ไม่สำคัญทิ้งจะไม่มีผลต่อการทำดัชนีและไม่ทำให้ประสิทธิภาพการค้นหาข้อมูลต่ำลงตัวอย่างคำที่ไม่สำคัญในภาษาอังกฤษ เช่น a, an, the, in, out, before, up, down เป็นต้น เนื่องจากคำที่ไม่สำคัญในภาษาอังกฤษมีการระบุไว้อย่างชัดเจนว่ามีคำอะไรบ้างในรายการที่เรียกว่า Stop Word List แต่ภาษาไทยยังไม่มีผู้ทำการคิดวิธีในการตรวจหาคำที่ไม่สำคัญในเอกสาร ตัวอย่างแนวคิดในการหาคำที่ไม่สำคัญในเอกสารนั้นสามารถอ่านรายละเอียดได้ใน (Lili H. & Lizhu H., 2008) ซึ่งเป็นตัวอย่างการหาคำที่ไม่สำคัญในภาษาจีนซึ่งนักวิจัยอาจจะนำมาประยุกต์ใช้ได้

1.5 ทำการดัชนีข้อมูลโดยใช้เทคนิคต่างๆ เช่น วิธีดัชนีแบบอินเวิร์ท ซึ่งประกอบด้วย พจนานุกรม (Dictionary) และการประกาศ (Posting) ดิกชันนารีจะเก็บไว้ในหน่วยความจำหลักและมีตัวชี้ (Pointer) ชี้ไปยังโพสติ้งต่างๆ ที่เก็บไว้บนฮาร์ดดิสก์ คำสำคัญจะถูกเรียงลำดับตามตัวอักษรพร้อมกับความถี่ของคำ คำนั้นที่ปรากฏในเอกสารต่างๆ คำเดียวกันจะนับความถี่รวมกันส่วนนี่เรียกว่า “ดิกชันนารี” และมีตัวชี้ชี้ไปยังโพสติ้งของแต่ละคำซึ่งโพสติ้งจะเก็บรายการเอกสารต่างๆ ที่คำนั้นๆ ปรากฏ

ขั้นตอนสำคัญของการทำดัชนีแบบอินเวิร์ทนี้คือ การเรียงลำดับข้อมูล ซึ่งโดยทั่วไปจะเรียงลำดับตามตัวอักษร (Alphabet Sorting) คำเดียวกันที่ปรากฏอยู่ในเอกสารต่างกันจะถือเป็นหนึ่งคำและจะนำ

จำนวนครั้งหรือความถี่ของคำนั้นๆ ที่ปรากฏในทุกเอกสารมารวมกัน การทำดัชนีแบบอินเวิร์ทประกอบด้วยส่วนสำคัญ 2 ส่วนคือ 1) ดิกชันนารีคือส่วนที่ทำหน้าที่ในการบันทึกข้อมูลสถิติ เช่น จำนวนเอกสารทั้งหมดที่คำนั้นๆ ปรากฏ หรือเรียกว่า ความถี่เอกสาร (Document Frequency) ข้อมูลนี้จะถูกนำพิจารณาเพื่อช่วยในการเรียงลำดับข้อมูลผลลัพธ์ด้วยเช่นกัน 2) โพสติ้ง คือ ส่วนที่แสดงรายการเอกสารทั้งหมดที่คำนั้นๆ ปรากฏโดยแสดงข้อมูลในรูปแบบของหมายเลขเอกสาร (Document ID) ดังแสดงในภาพที่ 2.2 การดัชนีแบบอินเวิร์ทนี้ถือเป็นวิธีการสำคัญและมีประสิทธิภาพสำหรับการดัชนีเอกสารที่เป็นตัวอักษร นอกจากการทำดัชนีแบบอินเวิร์ทแล้วยังมีวิธีการทำดัชนีอื่นๆ ที่มีประสิทธิภาพ ได้แก่ การทำดัชนีจากตำแหน่งคำ (Positional Indexing)



รูปที่ 2.2 แสดงส่วนประกอบหลักของการทำดัชนีแบบอินเวิร์ท (Inverted Indexing)

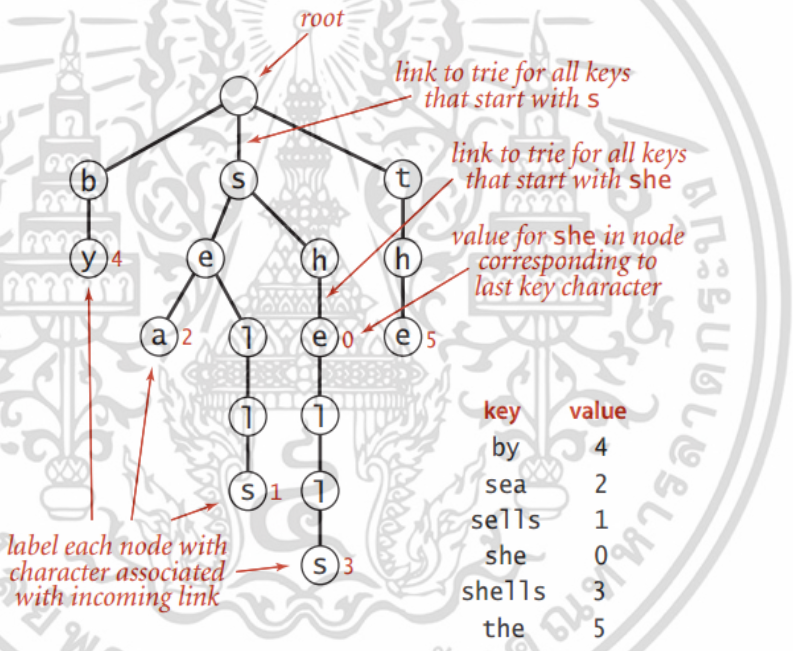
2.1.2 ทรัย (Trie)

โครงสร้างข้อมูลที่สร้างขึ้นจากตัวอักษรของคีย์สตริง ช่วยให้สามารถใช้ตัวอักษรมาใช้ในการค้นหาได้ ทรัย [8] เป็นชื่อที่ถูกเรียกโดย E. Fredkin เมื่อปี 1960 เนื่องจากเป็นโครงสร้างข้อมูลที่ใช้สำหรับการเรียกข้อมูล ออกเสียงว่า “try” เพื่อหลีกเลี่ยงความสับสนกับ “tree”

1. คุณสมบัติพื้นฐาน (Basic properties)

ทรัยเป็นโครงสร้างข้อมูลที่ประกอบด้วยโหนดที่ว่าง (null) หรือไม่ก็โหนดที่มีการอ้างอิงไปยังโหนดอื่นๆ โหนดแต่ละโหนดจะชี้ไปเพียงแค่หนึ่งโหนด ซึ่งเรียกว่า โหนดแม่ (ยกเว้นโหนดราก ที่ไม่มีโหนดใดชี้ไปที่มัน) และแต่ละโหนดจะมีลิงก์ (link) R ซึ่ง R คือ ขนาดของตัวอักษร ทรัยมีลิงก์ว่างเป็นจำนวนมาก ดังนั้น เมื่อเราวาดทรัยเรามักจะละเว้นลิงก์ว่างแม้ว่าจะมีลิงก์ที่ชี้ไปยังโหนดอื่นได้ ลิงก์แต่ละลิงก์นั้นมีความสัมพันธ์กับค่าตัวอักษร เพราะแต่ละจุดมีการเชื่อมโยงไปยังโหนดอื่นได้ โหนดแต่ละโหนดยังมีค่าที่สอดคล้องกัน ซึ่งอาจจะเป็นค่าว่างหรือค่าที่มีความสัมพันธ์กับคีย์ของสตริงในตารางสัญลักษณ์ โดยเฉพาะเราจะเก็บค่าที่สัมพันธ์กับคีย์แต่ละคีย์ไว้ในโหนดที่สอดคล้องกับตัวอักษรตัวสุดท้ายของมัน

ตัวอย่าง โครงสร้างข้อมูลแบบทรัย แสดงดังรูปที่ 2.3 ต่อไปนี้



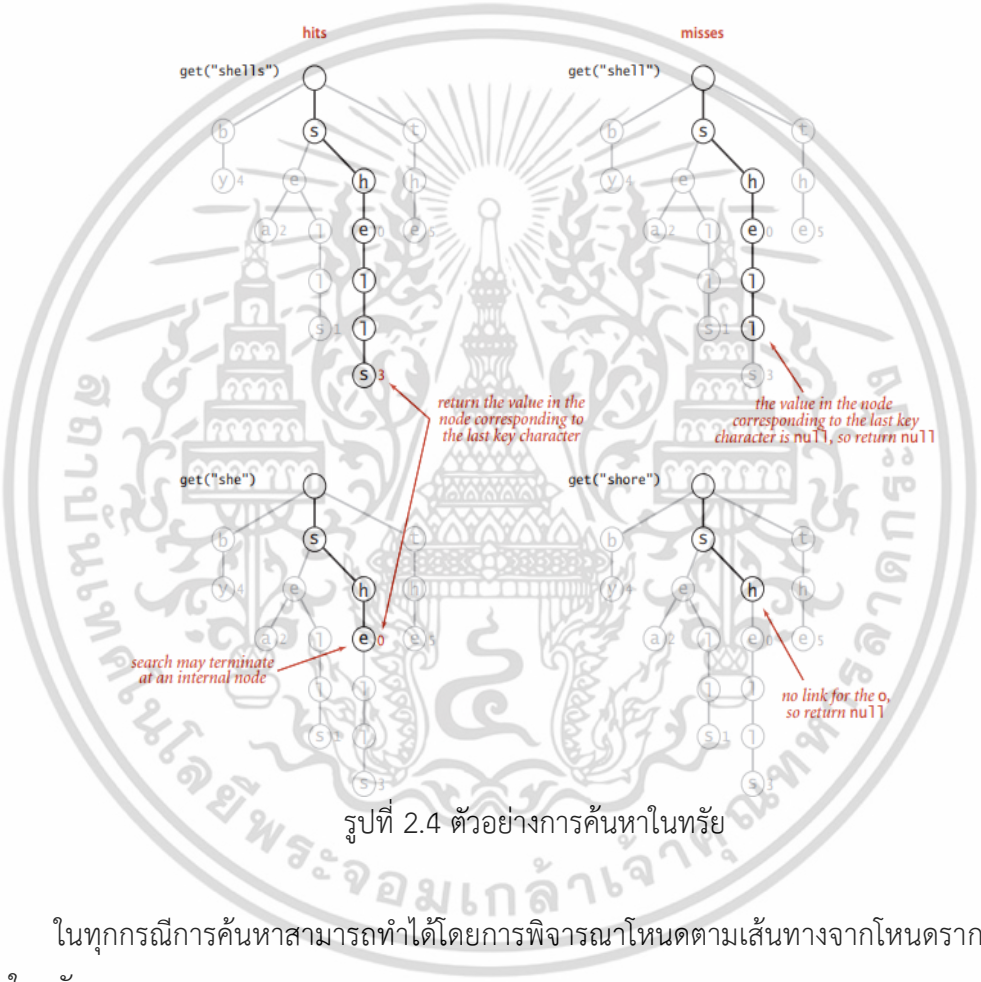
รูปที่ 2.3 โครงสร้างข้อมูลแบบทรัย

2. การค้นหาในทรัย (Search in a trie)

การหาค่าที่เกี่ยวข้องกับคีย์สตริงที่กำหนดไว้ในทรัยเป็นกระบวนที่ง่ายโดยที่เราใช้ตัวอักษรเป็นคีย์ในการค้นหา แต่ละโหนดในทรัยมีลิงก์ที่สัมพันธ์กับตัวอักษรแต่ละสตริง เราจะเริ่มจากโหนดรากแล้วค้นหาลิงก์ที่เกี่ยวข้องกับตัวอักษรตัวแรกในคีย์นั้น จากโหนดข้างต้นเราจะตามลิงก์ที่เกี่ยวข้องกับตัวอักษรตัวที่สองแล้วต่อด้วยตัวที่สามไปเรื่อยๆ จนกว่าจะถึงตัวอักษรตัวสุดท้ายของคีย์หรือลิงก์ นั้นว่าง (ตัวอย่างการค้นหาแสดงดังรูปที่ 2.4) จากจุดนี้ต้องประกอบด้วยเงื่อนไขข้อใดข้อหนึ่งดังต่อไปนี้

- ค่าของโหนดที่สัมพันธ์กับตัวอักษรตัวสุดท้ายในคีย์ที่ไม่เป็นค่าว่างซึ่งเรียกว่า Search hit - ค่าที่สัมพันธ์กับคีย์ คือ ค่าในโหนดที่สอดคล้องกับตัวอักษรตัวสุดท้าย
- ค่าของโหนดที่สัมพันธ์กับตัวอักษรตัวสุดท้ายในคีย์ที่เป็นค่าว่างซึ่งเรียกว่า Search miss - คีย์ไม่มีอยู่ในตาราง
- การค้นหาที่สิ้นสุดลงด้วยลิงก์ว่างซึ่งจะเป็น search miss

ตัวอย่าง การค้นหาในทรัย แสดงดังรูปที่ 2.4 ต่อไปนี้



รูปที่ 2.4 ตัวอย่างการค้นหาในทรัย

ในทุกกรณีการค้นหาสามารถทำได้โดยการพิจารณาโหนดตามเส้นทางจากโหนดรากไปยังโหนดอื่นๆในทรัย

3. การเพิ่มข้อมูลในทรัย (Insertion into a trie)

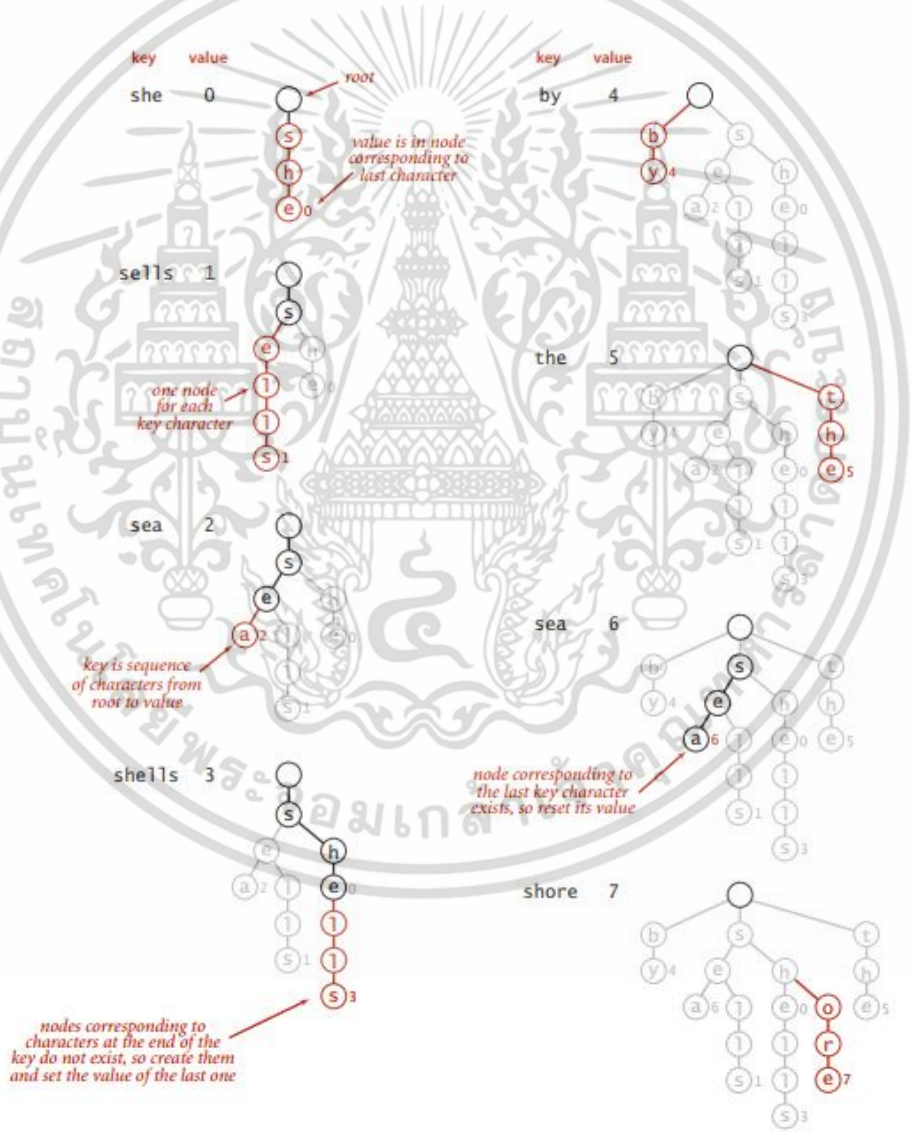
เช่นเดียวกับต้นไม้ค้นหาแบบทวิภาค (Binary search tree) เราจะทำการค้นหาก่อนการเพิ่มข้อมูล ในทรัยหมายถึงการใช้คีย์ตัวอักษรเพื่อเป็นแนวทางในการเพิ่มข้อมูลลงทรัย จนกระทั่งถึงตัวอักษรตัวสุดท้ายของคีย์หรือลิงก์ว่างโดยมีเงื่อนไขต่อไปนี้

- เราจะพบลิงก์ว่างก่อนที่จะถึงตัวอักษรตัวสุดท้ายของคีย์ ในกรณีนี้ไม่มีโหนดใดที่สอดคล้องกับตัวอักษรตัวสุดท้ายในคีย์ ดังนั้นเราจึงจำเป็นต้องสร้างโหนดสำหรับแต่ละตัวอักษรที่ยังไม่พบและทำการกำหนดค่าคีย์ตัวสุดท้ายที่เกี่ยวข้อง
- เราพบตัวอักษรตัวสุดท้ายของคีย์ก่อนที่จะถึงลิงก์ว่างในกรณีนี้เราจะกำหนดค่าของโหนดที่จะเชื่อมโยงกับคีย์และเก็บ associative array ด้วย

ในทุกกรณี เราจะวิเคราะห์หรือสร้างโหนดในทรัยสำหรับแต่ละคีย์ตัวอักษร

ตัวอย่าง การเพิ่มข้อมูลในทรัย แสดงดังรูปที่ 2.5 ต่อไปนี้

she sells sea shells by the sea shore

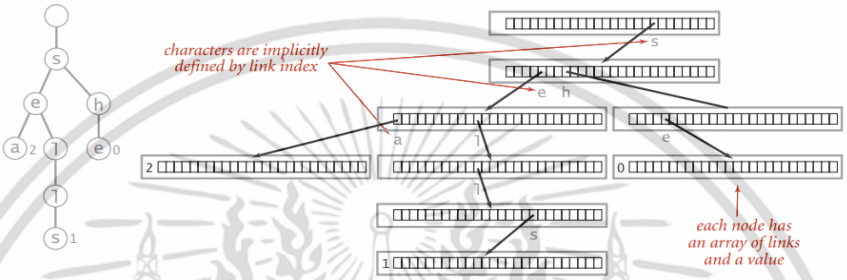


รูปที่ 2.5 การเพิ่มข้อมูลในทรัย

4. โหนดตัวแทน (Node representation)

แผนภาพของทรัยนั้นค่อนข้างไม่สอดคล้องกับโครงสร้างข้อมูลของโปรแกรมที่เราจะสร้างขึ้น เพราะไม่มีการวาดลิงก์ว่างดังนั้น การนำลิงก์ว่างเข้าโครงสร้างจะต้องมีลักษณะสำคัญต่อไปนี้

- ทุกโหนดต้องมีลิงก์ R สำหรับแต่ละตัวอักษรที่เป็นไปได้
- ตัวอักษรและคีย์จะถูกเก็บไว้ในโครงสร้างข้อมูลโดยปริยาย



รูปที่ 2.6 Trie representation (R = 26)

จากรูปที่ 2.6 ที่แสดงให้เห็นทรัยสำหรับคีย์ที่สร้างขึ้นจากตัวอักษรตัวพิมพ์เล็กที่แต่ละโหนดมีค่าและมีภายในโหนด 26 ลิงก์ จุดแรกของลิงก์ใน subtrie สำหรับคีย์ที่เริ่มต้นด้วย a จุดที่สองสำหรับสตริงที่เริ่มต้นด้วย b เป็นต้น คีย์ในทรัยจะแสดงเส้นทางจากโหนดรากจนถึงโหนดสุดท้ายที่มีค่าที่ไม่ใช่ ค่าว่าง

ตัวอย่างเช่น คำว่า sea มีความสัมพันธ์กับค่า 2 ในทรัยเพราะลิงก์ที่ 19 ในโหนดราก (สำหรับคีย์ทั้งหมดที่เริ่มต้นด้วย s) ไม่เป็น null และลิงก์ที่ 5 ในโหนดถัดไปไม่เป็น null (สำหรับคีย์ทั้งหมดที่เริ่มต้นด้วย se) และลิงก์แรกในโหนดมีค่าเท่ากับ 2 ไม่ใช่คำว่า sea หรือตัวอักษร s, e และ a จะถูกเก็บไว้ในโครงสร้างข้อมูล เพราะแท้จริงแล้วโครงสร้างข้อมูลไม่ได้ประกอบด้วยอักษรหรือสตริงเพียงแต่มีลิงก์และ value เท่านั้น เนื่องจากพารามิเตอร์ R มีบทบาทที่สำคัญ เราจึงเรียกทรัยสำหรับตัวอักษร R-character เป็น R-way trie แสดงได้ดังรูปที่ 2.6

2.1.3 Serialization

ภาษา Java มี streams อยู่ 3 ชนิด คือ byte streams, char streams และ object streamใน ที่นี้จะกล่าวถึงเฉพาะ object streams ซึ่งเป็น streams ที่จะถูก read/write เป็น object ใน package java.io มี

- คลาส ObjectOutputStream มี method

public final void writeObject(Object) throws IOException

สำหรับ write ค่า value ของ object นั้นออกไปทาง stream ที่ต่ออยู่

- คลาส ObjectInputStream มี method

public final Object readObject() throws IOException, ClassNotFoundException สำหรับ read ค่า(ที่เป็น binary)ใน stream ออกมาเป็น value ของ object นั้น

- Interface ที่ชื่อว่า Serializable เป็น tag interface ที่ไม่มีสมาชิก

writeObject() ใช้ทำ Serialization [7] คือการนำค่า value ของ object หนึ่ง ออกมายืดเป็นเส้น (เป็น array ของ bytes) เพื่อส่งออกไปทาง stream ได้ และใช้ readObject() ทำ De-serialization คือการอ่านค่าของ object ออกมาจาก stream เพื่อนำไปใช้งาน

หมายเหตุ คลาสที่จะถูกทำ Serialization ได้ ต้องเป็นคลาสที่ implements Serializable

นั่นคือ writeObject() จะตรวจสอบก่อนว่า Object ที่เป็นพารามิเตอร์นั้น เป็น instance ของคลาสที่ implements Serializable หรือไม่ ถ้าใช่จะยอม write ค่า Object นั้นออกไปให้ แต่ถ้าไม่จะโยน NotSerializableException ออกมา ดังรูปที่ 2.7

```
import java.io.*;
class A implements Serializable {
    int x;
    A(int x) { this.x = x; }
}
class Serial {
    public static void main(String[] args) throws Exception {
        FileOutputStream fos = new FileOutputStream("tmp");
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(new A(123));
        oos.close();
    }
}
```

รูปที่ 2.7 implements Serializable

ในตัวอย่างนี้ (แสดงดังรูปที่ 2.7) instance ของคลาส A ที่มี x เป็น 123 จะถูกเก็บลงในไฟล์ tmp

Try:

1. หากคลาส A ไม่ implements Serializable จะเกิดอะไรขึ้น ตอนคอมไพล์โปรแกรมและตอนโปรแกรมทำงาน

- หากคลาส B เป็นลูกของคลาส A แต่คลาส B ไม่ implements Serializable จะถูก serialized ได้หรือไม่
- หาก interface extends จาก Serializable คลาสที่ implements จะถูก serialized ได้หรือไม่

2. ตรวจสอบไฟล์ tmp ว่ามีข้อมูลใดบ้าง แล้วบอกได้หรือไม่ว่า ค่าของ Object ถูกเก็บไว้อย่างไร

3. ตรวจสอบขนาดของไฟล์ tmp เปรียบเทียบกับไฟล์ A.class แล้วบอกได้หรือไม่ว่า ข้อมูลของคลาส A ในไฟล์ A.class ถูกเก็บลงไปในไฟล์ tmp หรือไม่ เราจะแสดงการทำ De-serialization ดังรูปที่ 2.8

```
class DeSerial {
    public static void main(String[] args) throws Exception {
        FileInputStream fis = new FileInputStream("tmp");
        ObjectInputStream ois = new ObjectInputStream(fis);
        A a = (A) ois.readObject();
        System.out.println(a.x);
        ois.close();
    }
}
```

รูปที่ 2.8 การทำ De-serialization

ในตัวอย่างนี้ (แสดงดังรูปที่ 2.8) จะทำการอ่านค่าของ instance ที่ถูกเก็บไว้ในไฟล์ tmp กลับมาเป็น A a ซึ่งเมื่อพิมพ์ค่า a.x ออกมาจะได้เป็น 123 ดังที่เก็บไว้นั่นเอง

- ข้อมูลของ instance ที่อยู่ในไฟล์ tmp นั้นจะมีชื่อของคลาสเก็บ writeObject() จะอ่านข้อมูลนี้แล้วสร้าง instance ของคลาสนั้นขึ้นมา(โดยไม่ใช้ constructor)แล้วจึงกำหนดค่า data members ให้(โดยไม่ได้ใช้ setter และถึงแม้ data นั้นจะ private ก็ตาม)
- เมื่อ instance หนึ่งถูกทำ serialization ค่า state (คือค่าของ data members) ของทั้ง instance นั้นและของคลาสจะถูก serialized
- การทำ de-serialization อาจเกิดขึ้นที่เวลาห่างกันอย่างมาก หรือเกิดขึ้นที่เครื่องต่างจากเครื่องที่ทำ serialization แต่จะต้องมีไฟล์ .class ของคลาสที่ถูก serialized
- เราสามารถทำ serialization ค่าของ instance หนึ่งใส่ลงใน stream ที่เป็นหน่วยความจำ แล้วทำ de-serialization ได้ออกมาเป็นอีก instance หนึ่งที่มี state เหมือนกัน ดังนั้น ถ้าคลาสใดถูก serialized ได้ ก็จะหมายความว่าคลาสนั้นจะถูก clone ได้

การที่ต้องให้ผู้พัฒนาโปรแกรมติด interface Serializable ให้แก่คลาส เพื่อยอมให้ถูก serialized ได้ ก็เพราะการทำ serialization ได้จะทำให้ไม่ปลอดภัยเหมือนกับการ clone ได้ นั่นเอง

หากคลาสมีสมาชิกเป็น instances เมื่อ instance ของคลาสนั้นถูกทำ serialization ค่าของ instances ที่เป็นสมาชิกทั้งหมดทุกระดับจะถูก serialized หมายความว่าการทำ serialization จะเป็นแบบ deep โดย default ดังรูปที่ 2.9

```
import java.io.*;
class A implements Serializable {
    B b;
    A(int x) { b = new B(x); }
}
class B implements Serializable {
    int x;
    B(int x) { this.x = x; }
}
class SerHier {
    public static void main(String[] args) throws Exception {
        FileOutputStream fo = new FileOutputStream("tmp");
        ObjectOutputStream os = new ObjectOutputStream(fo);
        os.writeObject(new A(123));
        os.close();

        FileInputStream fi = new FileInputStream("tmp");
        ObjectInputStream oi = new ObjectInputStream(fi);
        A a = (A) oi.readObject();
        System.out.println(a.b.x);
        oi.close();
    }
}
```

รูปที่ 2.9 implements Serializable

แต่คลาสของ instances ที่เป็นสมาชิก ต้อง implements Serializable ทั้งหมดแสดงดังรูปที่ 2.9

1. Transient

โดย default เมื่อทำ serialization ค่า state ของ instance (ทั้ง static และไม่ static) จะถูก serialized แต่บางกรณีเราอาจไม่ต้องการจะ serialize ค่าสมาชิกบางตัวของคลาสนั้น เนื่องจาก

- ค่าของสมาชิกตัวนั้น ไม่มีความจำเป็นต้องถูกส่งออกไป หรือเก็บไว้
- ค่าของสมาชิกตัวนั้นมีขนาดใหญ่มาก ซึ่งจะเสียเวลาในการส่ง และใช้หน่วยความจำเป็นจำนวนมาก
- มี instances บางประเภทที่ เมื่อทำ serialization ไว้แล้ว de-serialization กลับมา จะใช้งานได้ไม่เหมือนเดิม เช่น พวกที่เป็น Connection หรือ Session

กรณีเช่นนี้ เราไม่ควร serialize ค่าแบบนี้ และยอมให้ไม่มีค่า หลังจากทำ de-serialization แต่เราก็สามารถสร้างขึ้นใหม่ หรืออ่านค่าขึ้นมาใหม่ได้ดังรูปที่ 2.10

* ภาษา Java มี keyword 'transient' สำหรับวางไว้หน้าสมาชิกตัวที่เราไม่ต้องการให้ถูก serialized

ตัวอย่าง การทำ Transient แสดงดังรูปที่ 2.10

```
import java.io.*;
class A implements Serializable {
    int x;
    transient int y;
    A(int x, int y) { this.x = x; this.y = y; }
}
class Transient {
    public static void main(String args[]) throws Exception {
        A c = new A(1, 2);
        FileOutputStream fo = new FileOutputStream("tmp");
        ObjectOutputStream(fo);
        oo.writeObject(c);
        oo.close();

        FileInputStream fi = new FileInputStream("tmp");
        ObjectInputStream oi = new ObjectInputStream(fi);
        A m = (A) oi.readObject();
        System.out.println(m.x + "," + m.y);     // 1, 0
        oi.close();
    }
}
```

รูปที่ 2.10 การทำ Transient

Try:

1. ตรวจสอบขนาดของไฟล์ tmp เปรียบเทียบกับระหว่าง กรณีมี y ที่ transient กับไม่ transient
2. transient มีผลกับเฉพาะ instance state เท่านั้น แต่ไม่มีผลกับ class state หมายความว่า สมาชิกที่ static และ transient จะถูก serialized

2. Externalizable

ภาษา Java มี Externalizable เป็น interface ที่ extends จาก Serializable โดยเพิ่มสอง methods คือ

public void writeExternal(ObjectOutput) throws IOException;

public void readExternal(ObjectInput) throws IOException, ClassNotFoundException;

คลาสที่ implements Externalizable จะถูก serialized เหมือนกับคลาสที่ implements Serializable แต่เมื่อ instance ของคลาสนั้นถูก serialized โดย writeObject() จะทำให้ writeExternal() ของคลาสนั้นถูกทำงานและเมื่อถูก de-serialized โดย readObject() จะทำให้ readExternal() ของคลาสนั้นถูกทำงาน

แสดงเราสามารถกำหนดการทำ serialization และ de-serialization ของคลาสหนึ่งได้ โดยการ implements methods ดังกล่าว โดยปกติ เราจะทำเช่นด้วยสองเหตุผล คือ

- เพื่อเลือกว่าจะ serialized ค่าใดออกไปบ้าง และด้วยลำดับอย่างไร
- เพื่อเพิ่มการทำงานบางอย่างตอนทำ serialization และ de-serialization เช่นอาจมีการเข้ารหัสและถอดรหัส

ตัวอย่าง implements Exterrializable แสดงดังรูปที่ 2.11

```
import java.io.*;
class A implements Externalizable {
    private int x = 0;
    private String msg = null;
    public A() {  }
    public A(int x, String msg) { this.x = x; this.msg = msg; }
    public void writeExternal(ObjectOutput out) throws IOException {
        out.writeObject(msg);
        out.writeInt(encode(x));
    }
    public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
        msg = (String) in.readObject();
        x = decode(in.readInt());
    }
    public String toString() { return x + "," + msg; }
    private int encode(int x) { return ++x; }
    private int decode(int x) { return --x; }
}
class ExtSerial {
    public static void main(String[] args) throws Exception {
        FileOutputStream fos = new FileOutputStream("tmp");
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(new A(123, "Hello"));
        oos.close();

        FileInputStream fis = new FileInputStream("tmp");
        ObjectInputStream ois = new ObjectInputStream(fis);
        A a = (A) ois.readObject();
        System.out.println(a);
        ois.close();
    }
}
```

รูปที่ 2.11 implements Exterrializable

คลาสที่ implements Exterrializable จะต้องมี empty constructor ดังรูปที่ 2.11

3. Serial Version UID

Try:

1. หากเราสร้าง instance ของคลาส A ที่ implements Serializable และทำ serialization ไว้ในไฟล์ tmp ดังในโปรแกรม Serial.java ที่แสดงดังรูปที่ 2.12 นี้

```java
import java.io.*;
class A implements Serializable {
    int x;
    A(int x) { this.x = x; }
}
class Serial {
    public static void main(String[] args) throws Exception {
        FileOutputStream fos = new FileOutputStream("tmp");
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(new A(123));
        oos.close();
    }
}
```

รูปที่ 2.12 การทำ serialization

หากแก้ไขคลาส A โดย int y เข้าไปสักตัวหนึ่งแสดงดังรูปที่ 2.13 นี้

```java
class A implements Serializable {
    int x;
    int y;
    A(int x) { this.x = x; }
}
```

รูปที่ 2.13 แสดงการแก้ไขคลาส A

คอมไพล์ A.java นี้ได้เป็นไฟล์ A.class รุ่นใหม่ แล้วทำ de-serialization เพื่ออ่านค่า instance ที่เก็บไว้ในไฟล์ tmp ด้วยโปรแกรม DeSerial.java ดังรูปที่ 2.14 นี้

```java
class DeSerial {
    public static void main(String[] args) throws Exception {
        FileInputStream fis = new FileInputStream("tmp");
        ObjectInputStream ois = new ObjectInputStream(fis);
        A a = (A) ois.readObject();
        System.out.println(a.x);
        ois.close();
    }
}
```

รูปที่ 2.14 การทำ de-serialization

จะเห็นว่าเกิด exception ขึ้นตอนโปรแกรมทำงาน เนื่อง instance ที่เก็บไว้ในไฟล์ tmp นั้นเป็นของคลาส A.class รุ่นเก่า แต่เราใช้ A.class รุ่นใหม่มาทำ de-serialization ปัญหาอยู่ตรงที่ เวลาที่เราทำ serialization กับ de-serialization อาจห่างกันนานมาก จนกระทั่งคลาสที่ใช้ตอนทำ serialization อาจถูกเปลี่ยนรุ่นไปแล้ว ภาษา Java จึงต้องหาวิธีแก้ปัญหานี้

1. ทดลองนำคลาส A (ที่ implements Serializable) นี้ไปอยู่ใน Eclipse editor จะเห็นว่ามี warning ตรวจสอบว่าเป็นการ warn อะไร

ภาษา Java มีข้อกำหนดว่า คลาสที่ implements Serializable จะต้องมี serial version uid (ซึ่งเราจะเรียกสั้นๆว่า uid) โดยมีกฎเกฑณ์ ดังนี้

- ถ้าคลาสใดกำหนด uid มาเอง ก็จะมี uid ตามนั้น
- หากคลาสใดไม่กำหนด uid เอง คอมไพเลอร์จะคำนวณให้จากค่าของชื่อ และ modifiers ทั้งหมดที่ปรากฏในคลาส ยกเว้น body ของ methods และค่าที่กำหนดให้แก่ data members

2. ในไดเรกทอรี \bin ของ JDK มีโปรแกรม serialver สำหรับใช้ตรวจสอบ uid ของคลาสเปิด command prompt ไปที่ไดเรกทอรีที่มีไฟล์ A.class แล้วทดลอง

serialver A

- ทดลองเปลี่ยนแปลงคลาส A โดยเพิ่ม data หรือ method แล้วตรวจสอบ uid
- ทดลองเปลี่ยนแปลงคลาส A โดยกำหนดให้แก่ data members หรือเปลี่ยนแปลง body ของ methods แล้วตรวจสอบ uid

3. ภาษา Java จะยอมให้ ตอนทำ de-serialization ใช้คลาสที่ถูกเปลี่ยนแปลงไปแล้วจากตอนทำ serialization ได้ แต่ต้องมี uid เหมือนเดิม ทดลองใช้คลาส A ทำ serialization จากนั้นเปลี่ยนแปลงคลาส A ให้มี uid เหมือนเดิม แล้วใช้ทำ de-serialization จะเห็นว่าการทำเช่นนี้มีข้อจำกัดที่ เราเปลี่ยนแปลงคลาสเฉพาะค่าที่กำหนดให้แก่ data members และเปลี่ยน body ของ methods เท่านั้น เรามีวิธีที่ดีกว่า คือ การค่า uid เอง

4. เพิ่มบรรทัดข้างล่างนี้ในคลาส A

static final long serialVersionUID = 1234567L;

แล้วทดลองเพิ่มหรือลดสมาชิกในคลาส รวมทั้งเปลี่ยน modifiers เพื่อดูว่า uid จะเปลี่ยนหรือไม่ แต่ถึงแม้คลาสที่มี uid เหมือนเดิมจะใช้ทำ de-serialization ได้ แต่ถ้าคลาสใหม่นั้นทำงานได้ไม่เหมือนคลาสเดิม ก็จะเกิดความผิดพลาดตอนโปรแกรมทำงานได้ ดังนั้น ถ้าจะเปลี่ยนแปลงคลาส ควรทำเพียงแค่เปลี่ยน body ของ methods หรือเพิ่มสมาชิกเท่านั้น ส่วนสมาชิกเดิมเป็นเช่นใด อย่าเปลี่ยนแปลง หรือตัดทิ้ง

2.1.4 ความสัมพันธ์เวียนเกิด (Recurrence Relation)

ในวิชาคณิตศาสตร์ ความสัมพันธ์เวียนเกิด [9], [12] คือ สมการที่นิยามตัวเองจากลำดับก่อนหน้านั้น ประกอบด้วย เงื่อนไขเริ่มต้น (Initial condition หรือ Base case) และ นิยามความสัมพันธ์ที่มีการทำซ้ำ (Recursion) ตัวอย่างเช่น

จำนวนฟีโบนัชชี (Fibonacci numbers) คือจำนวนต่าง ๆ ที่อยู่ในลำดับจำนวนเต็มดังต่อไปนี้ 0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946 … (ลำดับ OEIS A000045)

โดยมีนิยามของความสัมพันธ์ว่า จำนวนถัดไปเท่ากับผลบวกของจำนวนสองจำนวนก่อนหน้า และสองจำนวนแรกก็คือ 0 และ 1 ตามลำดับ และลำดับของจำนวนดังกล่าวก็จะเรียกว่า **ลำดับฟีโบนัชชี**

หากเขียนให้อยู่ในรูปของสัญลักษณ์ ลำดับ F_n ของจำนวนฟีโบนัชชีนิยามขึ้นด้วยความสัมพันธ์เวียนเกิดดังนี้

$$F_n = F_{n-1} + F_{n-2}$$

โดยกำหนดเงื่อนไขเริ่มต้นให้

$F_0 = 0$

$F_1 = 1$

จำนวนฟีโบนัชชี (Fibonacci numbers) ลำดับที่ 5 เป็นดังนี้

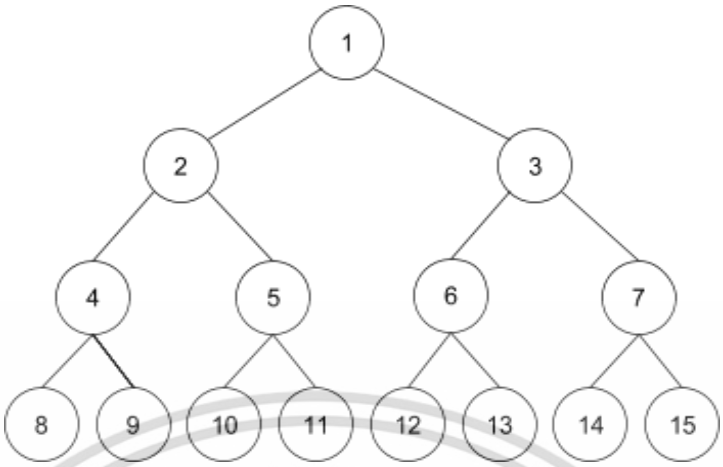
$F_5 = F_4 + F_3 = 3+2 = 5$

$F_4 = F_3 + F_2 = 2+1 = 3$

$F_3 = F_2 + F_1 = 1+1 = 2$

2.1.5 การค้นตามแนวกว้าง (Breadth first search)

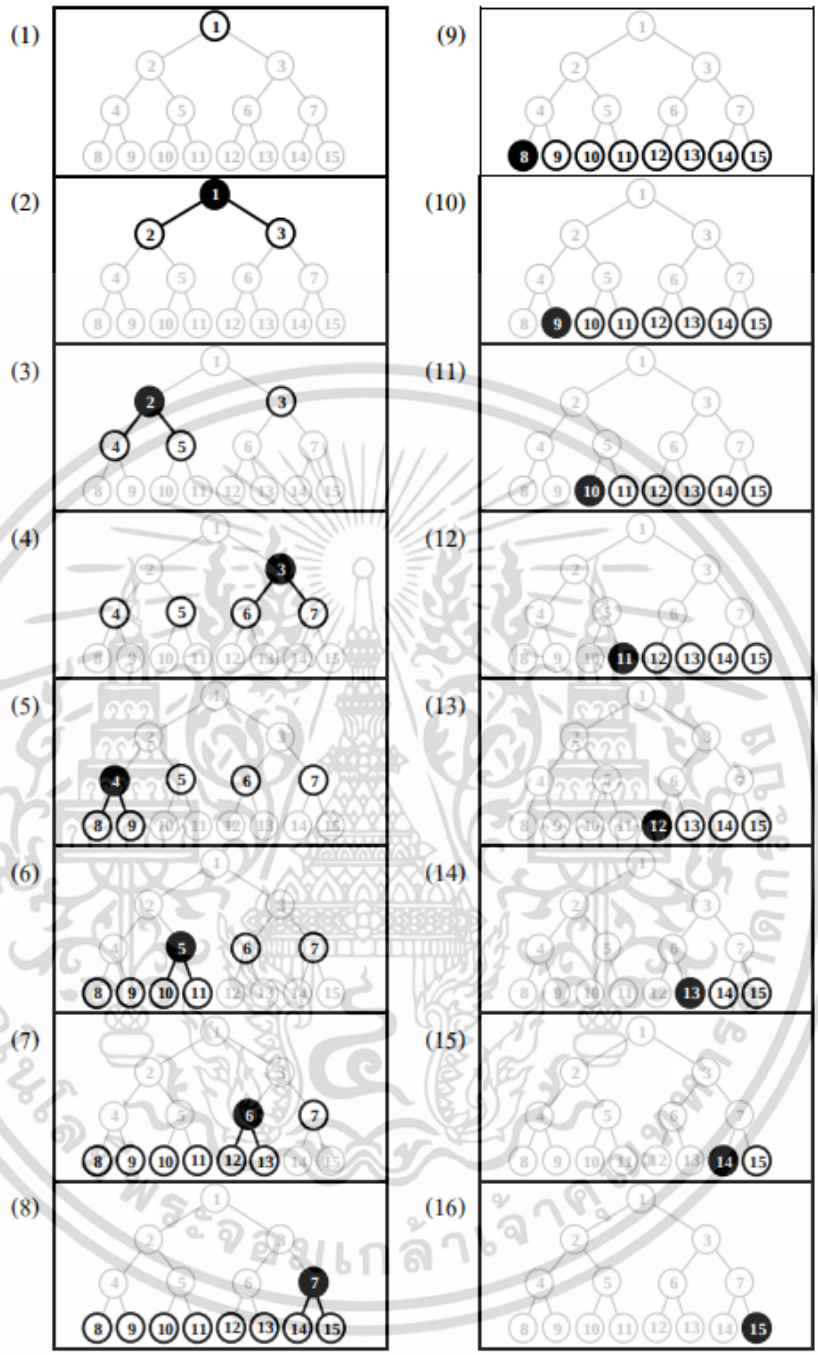
กระบวนการค้นคำตอบในต้นไม้ปริภูมิสถานะอีกแบบหนึ่งที่มีระบบระเบียบ คือ การค้นตามแนวกว้าง [5] ลักษณะการค้นเช่นนี้คล้ายการแวะผ่านจุดต่างๆ ในต้นไม้ทีละระดับ คือเริ่มตรวจสอบสถานะของจุดในระดับ 0 (ราก) ตามด้วยทุกๆ ในระดับที่ 1 ตามด้วยทุกๆ จุดในระดับที่ 2 ไล่ไปทีละระดับเช่นนี้จนถึงระดับล่างสุด ดังนั้น ลำดับการตรวจสอบสถานะของจุดของต้นไม้ในรูปที่ 2-13 ก็คือ 1, 2, 3, ..., 15 **ตัวอย่าง** ต้นไม้ปริภูมิสถานะ แสดงดังรูปที่ 2.15



รูปที่ 2.15 ตัวอย่างต้นไม้ปริภูมิสถานะ

การค้นตามแนวกว้างจะเป็นวงวนของการเลือก E–node มาหนึ่งจุดจาก live node ต่างๆ จากนั้นจะแตกกิ่ง E–node ที่เลือกมานี่เพื่อผลิตจุดลูกออกมาเป็น live node ให้หมดทุกลูก (แล้วตัวเองก็ถูกทำลายไป) ประเด็นนี้จะเห็นว่าต่างจากการค้นตามแนวลึกที่แตกกิ่งทีละกิ่งแล้วเปลี่ยนจุดที่เพิ่งถูกสร้างเป็น E–node ทันที นั่นคือในการค้นตามแนวลึกจุดใดเกิดก่อนตายทีหลัง แต่สำหรับการค้นตามแนวกว้างนั้น เกิดก่อนตายก่อน รูปที่ 2.16 แสดงตัวอย่างการค้นตามแนวกว้างในต้นไม้ปริภูมิสถานะในรูปที่ 2.15 ก่อนอื่นเราต้องตรวจสอบสถานะของจุดเมื่อจุดนั้นถูกสร้างเป็น live node และเราแตกกิ่งจุดเมื่อจุดนั้นถูกเลือกเป็น E–node การค้นเริ่มด้วยการให้รากของต้นไม้เป็น live node (แสดงด้วยวงกลมขาว – รูป 1) จากนั้นเปลี่ยนเป็น E–node (แสดงด้วยวงกลมดำ) แตกกิ่งสร้างจุดลูกเรียงลำดับจากซ้ายไปขวาได้ live node ใหม่ๆ แล้วตัวเองก็ถูกทำลายไป (รูป 2) จากนั้นลงมาอีกหนึ่งระดับ ไล่หยิบ live node จากซ้ายไปขวามาเปลี่ยนเป็น E–node แล้วแตกกิ่งได้ live node เพิ่มขึ้นอีกในระดับถัดไป กระทำเช่นนี้ไปเรื่อยๆ (ดูรูปประกอบ) จะเห็นว่าจุดต่างๆ ในต้นไม้ถูกสร้างขึ้นเป็น live node ตามลำดับการค้นตามแนวกว้าง

การค้นหาตามแนวกว้าง (Breadth first search) แสดงดังรูปที่ 2.16



รูปที่ 2.16 การค้นหาตามแนวกว้าง (Breadth first search)

ถ้าสังเกตลำดับของจุดในต้นไม้ที่ถูกสร้างขึ้นและลำดับของ live node ต่างๆ ที่ถูกเลือกมาเป็น E–node เพื่อแตกกิ่งต่อนั้นจุดที่ถูกสร้างเป็น live node ก่อนก็จะถูกเลือกออกมาแตกกิ่งก่อน ดังนั้นจึงเป็นธรรมชาติอย่างยิ่งที่เราจะจัดเก็บ live node ต่างๆ ไว้ใน queue ดังนั้น วงวนการทำงานระหว่างการ

ค้นตามแนวกว้าง คือการดึงจุดออกจาก queue เพื่อแตกกิ่งสร้างจุดลูกทั้งหมดเป็น live node เพิ่มใส่เข้าไปใน queue หมุนวนทำงานในลักษณะนี้จน queue หมด

ตัวอย่าง จงเขียนรหัสเทียมเพื่อค้นคำตอบของปัญหาผลรวมของเซตย่อยแบบตามแนวกว้าง กำหนดให้ A คือเซตของจำนวน และให้ P เป็นผลรวมที่ต้องการ

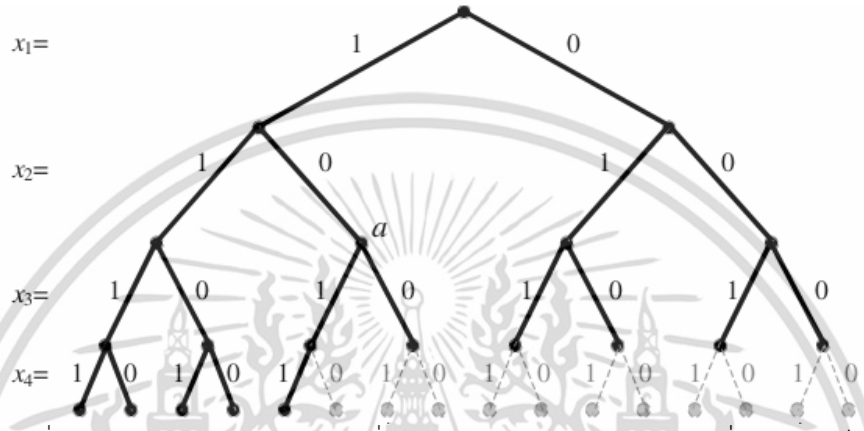
ถ้าเรากำหนดรูปแบบผลเฉลยที่เป็นเซตย่อยให้เป็นสตริงของบิตความยาว n เพื่อระบุว่าจะเลือกหรือไม่เลือกสมาชิกในเซตมาเป็นคำตอบ การค้นตามแนวกว้างก็จะเริ่มด้วยการสร้างราก ตรวจสอบราก เพิ่มรากเข้า queue จากนั้น เข้าวงวนเพื่อลบผลเฉลย x[1..k] กำกับจุดออกจาก queue เพื่อนำมาขยายโดยการเติมคำของ x[k+1] ตรวจสอบ x[1..k+1] และเพิ่มเข้า queue ด้วยทุกๆ คำที่เป็นไปได้ของ x[k+1] แล้วกลับไปทำงานของวงวนต่อไปจนกว่า queue จะหมด เขียนได้เป็นรหัสเทียม แสดงดังรูปที่ 2.17

```
01: SumofSubset_BFS( A[1..n], P )
02: {
03:     Q = empty queue
04:     Check( A[1..n], P, x[1..0])
05:     enqueue( Q, x[1..0] );
06:     while ( isNotEmpty( Q ) ) {
07:         x[1..k] = dequeue( Q );
08:         if ( k < n ) {
09:             x[k+1] = 1;
10:             Check( A[1..n], P, x[1..k+1])
11:             enqueue( Q, x[1..k+1] );
12:             x[k+1] = 0;
13:             Check( A[1..n], P, x[1..k+1])
14:             enqueue( Q, x[1..k+1] );
15:         }
16:     }
17: }
18: ---------------------------------------------------
19: Check( A[1..n], P, x[1..k] )
20: {
21:     if ( k==n AND       A [ i ] x [ i ] == P )
22:         print( A[1..n], x[1..n] )
23: }
```

รูปที่ 2.17 รหัสเทียมค้นคำตอบของปัญหาผลรวมของเซตย่อยแบบตามแนวกว้าง

ให้สังเกตว่าจำนวน live node ที่มีอยู่ ณ ขณะใดขณะหนึ่งระหว่างการค้นตามแนวกว้างนั้นมีมากกว่ากรณีของการค้นตามแนวลึก เนื่องจากจำนวน live node ของการค้นตามแนวลึกแปรตามสูงของต้นไม้แต่จำนวน live node ของการค้นตามแนวกว้างแปรตามจำนวนจุดในแต่ละระดับ และต้นไม้ปริภูมิ

สถานะมักเป็นต้นไม้อ้วน จึงมีจำนวนจุดในแต่ละระดับมาก ถ้าหากจุดที่เป็นสถานะผลเฉลยมาอยู่ที่ใบหมด การค้นตามแนวกว้างก็ต้องแวะผ่านจุดภายในเป็นจำนวนมาก ตัวอย่างเช่น ต้องการหาเซตย่อยของ {25, 10,9, 2} ที่มีผลรวมเท่ากับ 36 พบว่ากว่าที่การค้นตามแนวกว้างจะพบคำตอบ ก็ต้องแวะผ่านจุดจำนวน 20 จุด ในขณะที่เราแวะผ่านเพียง 12 จุด ถ้าค้นตามแนวลึก (ดูตัวอย่างรูปที่ 2.18 ประกอบ)



รูปที่ 2.18 การค้นตามแนวกว้างเพื่อหาเซตย่อยของ {25,10,9,2} ที่มีผลรวมเป็น 36

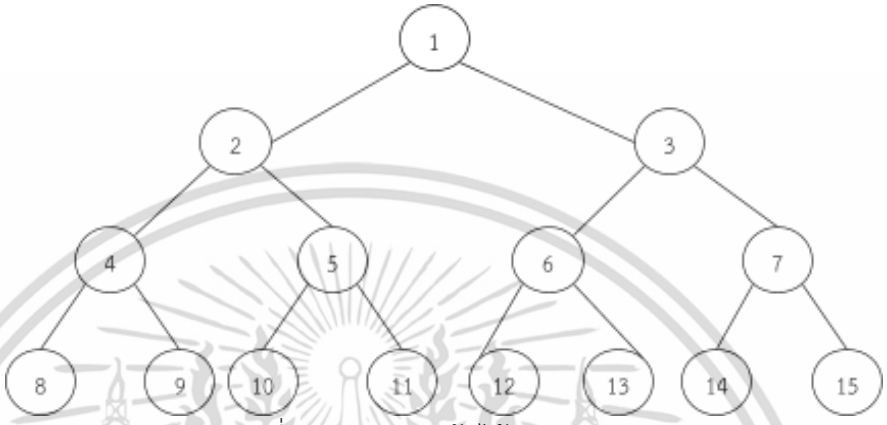
ค้นตามแนวกว้างมีข้อดี สำหรับกรณีที่จุดทั้งหลายในต้นไม้มีสถานะเป็นผลเฉลยนั้น การค้นตามแนวกว้างเมื่อพบจุดที่เป็นสถานะคำตอบแล้ว จะประกันได้ว่าพบจุดที่เป็นสถานะคำตอบที่อยู่ในระดับตื้นที่สุดของต้นไม้ (เพราะพิจารณาลงมาทีละระดับๆ) แล้วประเด็นนี้ข้อดีจะขึ้นกับตัวปัญหา ตัวอย่างเช่น ต้องการหาเซตย่อยที่ มีผลรวมเท่ากับค่าที่ต้องการ และต้องการให้ได้เซตย่อยขนาดเล็กที่สุดด้วย การค้นตามแนวกว้างจึงพิจารณาเซตย่อยขนาดเพิ่มขึ้นทีละหนึ่งทุกๆ แบบ ตามระดับที่กำลังค้น นอกจากนี้ สำหรับบางปัญหาเราอาจ พบต้นไม้ปริภูมิสถานะที่สูงมากๆ (หรือบางครั้งเป็นอนันต์) ถ้าคำตอบที่ค้นอยู่ทางขวาของต้นไม้ในระดับไม่ไกลนัก การค้นตามแนวลึกก็จะเสียเวลามากๆ ลงตามแนวลึก ถ้าเป็นต้นไม้อนันต์ก็จะติดในวงวน (จนในที่สุดข้อมูลล้น stack) ในขณะที่การค้นตามแนวกว้างจะค้นลงมาทีละระดับๆ และสามารถค้นพบคำตอบ

2.1.6 การค้นตามแนวลึก (Depth first search)

การค้นตามแนวลึก [5] เหมือนกับการแวะผ่านต้นไม้แบบก่อนลำดับคือ จะลงลึกไปเรื่อยๆ เริ่มด้วยการสร้างรากให้เป็นปมเป็น ปมนี้ถูกเลือกให้เป็นปมขยาย แตกกิ่งได้ปมเป็นปมใหม่หนึ่งปม ปมใหม่นี้ก็กลายเป็นปมขยายทันที เพื่อแตกกิ่งต่อได้ปมเป็นปมใหม่ ปมใหม่นี้ก็กลายเป็นปมขยายทันทีเพื่อแตกกิ่งต่ออีก เป็นเช่นนี้เรื่อยๆ จนเมื่อใดที่ปมขยายปัจจุบันแตกกิ่งหมดแล้ว (หรือแตกไม่ได้เพราะเป็นใบ) ก็จะถูก

ทำลาย และเลือกปมเป็นที่เกิดขึ้นล่าสุดก่อนหน้านี้เป็นปมขยายเพื่อแตกกิ่ง สรุปว่า ปมล่างจะแตกกิ่งเสร็จก่อนปมบน (นั่นคือ ปมที่เกิดทีหลังตายก่อน)

สมมติว่า เราต้องการค้นตามแนวลึกในต้นไม้ปริภูมิสถานะที่แสดงในรูปที่ 2.19

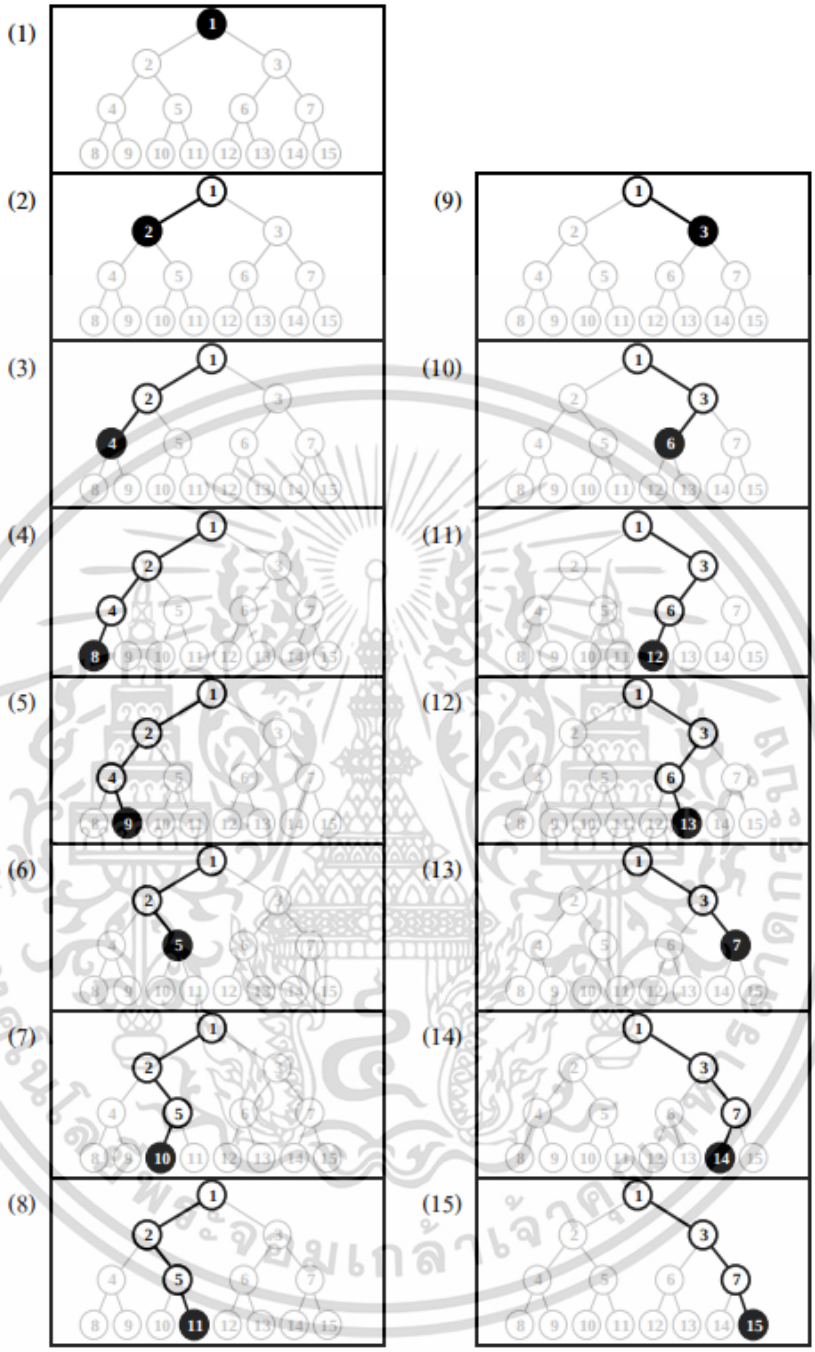


รูปที่ 2.19 ตัวอย่างต้นไม้ปริภูมิสถานะ

รูปที่ 2.19 แสดงปมต่างๆ ที่ถูกสร้างขึ้นระหว่างการค้นตามแนวลึกในต้นไม้นี้ วงกลมขาวคือปมเป็น ในขณะที่วงกลมดำคือปมขยาย เราตรวจสอบสถานะของปม ตอนที่ ถูกสร้างเป็นปมเป็น และเราแตกกิ่งปม เมื่อ ถูกเลือกเป็นปมขยาย เริ่มด้วยการสร้างรากที่ปม 1 แตกกิ่งซ้ายได้ปม 2 (รูป 2) ซึ่งก็แตกกิ่งซ้ายได้ปม 4 ต่อ (รูป 3) ซึ่งก็แตกกิ่งซ้ายได้ปม 8 (รูป 4) ถึงตรงนี้ปม 8 เป็นใบไม่แตกกิ่ง ก็ถูกทำลายไป ย้อนกลับขึ้นมาที่ปม 4 แตกกิ่งขวาได้ปม 9 (รูป 5) ซึ่งก็ถูกทำลายอีกเช่นกันเพราะเป็นใบ ย้อนกลับมาที่ปม 4 ซึ่งแตกกิ่งหมดแล้วก็ถูกทำลาย ย้อนกลับขึ้นมาที่ปม 2 แตกกิ่งขวาได้ปม 5 (รูป 6) กระทำการสร้างแตกกิ่ง และทำลายปม ในลักษณะเช่นนี้จนครบทุกปม (ดูรูป 2.20 ประกอบ) จะเห็นว่า ปมต่างๆในต้นไม้ถูกสร้างขึ้นเป็นปมเป็นตามลำดับการค้นตามแนวลึก

ข้อสังเกตที่น่าสนใจคือ จำนวนปมเป็นของต้นไม้ที่มีอยู่ ณ ขณะใดขณะหนึ่งนั้นเป็น O (h) โดยที่ h คือความสูงของต้นไม้ ซึ่งมีปริมาณน้อยมากๆ เมื่อเทียบกับขนาดของต้นไม้

การค้นหาตามแนวลึก (Depth first search) แสดงดังรูปที่ 2.20



รูปที่ 2.20 การค้นหาตามแนวลึก (Depth first search)

ตัวอย่าง จงเขียนรหัสเทียมเพื่อแก้ปัญหาผลรวมของเซตย่อยด้วยการค้นคำตอบแบบตามแนวลึก กำหนดให้ A คือเซตของจำนวน และให้ P เป็นผลรวมที่ต้องการ

ขอเลือกการแทนเซตย่อยด้วยอาเรย์ของ 0/1 เพื่อระบุว่า จะไม่เลือกหรือเลือกสมาชิกในเซตมาเป็นคำตอบ การแจงเซตย่อย EnumSubsets01 มีลักษณะการค้นตามแนวลึกอยู่แล้ว เพราะใช้การเรียกฟังก์ชันแบบซ้ำ มีลักษณะการเรียกซ้ำๆ ซ้อนกันไปเรื่อยๆ การเรียกครั้งหลังสุดจะคืนการทำงานก่อน จึงนำ EnumSubsets01 มาปรับเพิ่มให้ตรวจสอบว่าเซตย่อยที่แจกแจงมาได้นั้นมีผลรวมเท่ากับค่าที่ต้องการหรือไม่ เขียนได้เป็นรหัสเทียมแสดงดังรูปที่ 2.21

```
01: SumofSubset_DFS01( A[1..n], P, x[1..n], k )
02: {
03:     if( k==n ) {
04:         if( sum(i=1 --> n) A[i]x[i] == P) Print ( A, x )
05:     } else {
06:         x[k+1] = 1
07:         SumofSubset_DFS01( A, P, x, k+1 )
08:         x[k+1] = 0
09:         SumofSubset_DFS01( A, P, x, k+1 )
10:     }
11:}
```

รูปที่ 2.21 รหัสเทียมค้นคำตอบของปัญหาผลรวมของเซตย่อยแบบตามแนวลึก

ตัวฟังก์ชัน SumofSubset_DFS01 นอกจากจะรับอาเรย์ x และค่า k เหมือนกับการแจงเซตย่อย EnumSubsets01 ยังรับอาเรย์ A และค่า P ที่เป็นข้อมูลขาเข้าของปัญหาด้วย บรรทัดที่ 4 ทำงานนเมื่อได้เติมค่าในอาเรย์ x ครบทุกช่องแล้ว จึงตรวจสอบว่า ผลรวมของเซตย่อยที่กำหนดใน x นั้นมีค่าเท่ากับ P ตามที่ต้องการหรือไม่การทำงานยังคงดำเนินการต่อ ถึงแม้ว่าจะพบคำตอบหนึ่งแล้วก็ตาม หมายความว่า SumofSubset_DFS01 จะแสดงเซตย่อยทุกเซตที่มีผลรวมเป็น P หากใครไม่ชอบเขียนแบบเรียกซ้ำ ก็สามารถใช้กองซ้อนช่วยเก็บสถานะเอง (ซึ่งก็คือ ผลเฉลย x[1..k] ขอเก็บสถานะด้วยอาเรย์ k ช่อง เมื่อเป็นสถานะที่แทนผลเฉลยที่เติมค่าของ x แล้วตั้งแต่ช่องที่ 1 ถึง k) เหตุที่ใช้กองซ้อนเพราะว่า การค้นตามแนวลึกนั้นเลือกปมเป็นล่าสุดมาเป็นปมขยาย เพื่อให้มีพฤติกรรมเหมือนกับSumofSubset_DFS01 ที่เรียกซ้ำกรณีให้ค่า x[k+1] เป็น 1 ก่อนกรณีเป็น 0 เมื่อเราใช้กองซ้อน ก็ต้อง push ปมที่แทนกรณี x[k+1] เป็น 0 ก่อนกรณีเป็น 1 เพราะกรณีเป็น 1 จะได้ถูก pop ออกมาเป็นปมขยายทันทีในรอบหน้า (ในที่นี้ให้นึกไว้ว่าการ push ไม่ใช่การแตกกิ่ง แต่การแตกกิ่งจะเกิดขึ้นตอน pop) ดังนั้นการตรวจสอบสถาะของปมจะเกิดขึ้นหลัง pop การค้นตามแนวลึกจึงประกอบด้วยวงวนของการ pop ผลเฉลยที่กำกับปมออกมาตรวจสอบ ถ้าต้องขยายผลเฉลยต่อ ก็สร้างอาเรย์ใหม่ขนาดใหญ่ขึ้นอีก 1 และทำสำเนาอาเรย์ x ที่ pop

ออกมา (ด้วย CopyOf) แล้วให้ค่ากับช่องที่ k+1 และ push ผลเฉลยใหม่ ทำเช่นนี้กับทุกค่าที่ต้องเติมในช่องที่ k+1 จากนั้นกลับไปทำงานในวงวนเพื่อ pop ต่อไป โดยเริ่ม push สถานะแรกของรากที่เป็นอาเรย์ขนาด 0 ช่อง

2.1.7 การขยายและจำกัดเขต (Branch and bound)

การค้นคำตอบด้วยวิธีการย้อนรอย มีลักษณะการค้นตามแนวลึกโดยจะลงไปค้นต่อตามแนวลึกเฉพาะกับปมที่มีแววเท่านั้น จะเล็มต้นไม้ออกเมื่อพบปมที่ไม่มีแวว ประสิทธิภาพการทำงานจึงขึ้นกับวิธีการตรวจสอบความมีแววของปม การค้นตามแนวลึกนั้นเลือกลงไปทางกิ่งซ้ายก่อน รอจนย้อนรอยเสร็จกลับมาที่เดิมแล้วจึงแตกกิ่งค้นต่อ กระทำเช่นนี้จากซ้ายไปขวา เพราะเป็นระเบียบ เขียนโปรแกรมกะทัดรัดและใช้เนื้อที่หน่วยความจำไม่มาก (แปรตามความลึกของต้นไม้) แต่ถ้าต้องการให้ค้นพบคำตอบได้เร็วขึ้น ให้พิจารณาใช้การค้นอีกวิธีที่เรียกว่า การขยายและจำกัดเขต

การขยายและจำกัด [5] เขตเหมาะกับปัญหาการหาค่าเหมาะที่สุด โดยอาศัยการคำนวณขอบเขตล่างของต้นทุน* กำกับปมเป็นต่างๆแล้วใช้ขอบเขตนี้เป็นปัจจัยในการเล็มต้นไม้

ตัวอย่าง การใช้ขอบเขตของผลเฉลยเพื่อเล็มต้นไม้ แสดงดังรูปที่ 2.22

รูปที่ 2.22 ตัวอย่างการใช้ขอบเขตของผลเฉลยเพื่อเล็มต้นไม้

---

* - หรือเป็นขอบเขตบนของกำไรก็ได้ ทั้งนี้ขึ้นกับลักษณะของฟังก์ชันจุดประสงค์ว่า ต้องการค่าน้อยที่สุดหรือมากที่สุด แต่ในการบรรยายนี้จะขอสมมติให้เราสนใจหาต้นทุนน้อยสุด

ขอยกตัวอย่างให้เห็นแนวทางการค้นแบบขยายและจำกัดเขต ก่อนอื่นพิจารณาต้นไม้ปริภูมิสถานะในรูปที่ 2.22

- วงกลมขาวแทนปมในต้นไม้ที่แตกกิ่งเสร็จหมดแล้ว
- วงกลมเหลืองแทนปมเป็นที่ยังไม่ได้แตกกิ่ง
- สี่เหลี่ยมแทนปมที่เป็นสถานะผลเฉลยที่สมบูรณ์
- จำนวนภายในสี่เหลี่ยมคือ ต้นทุนจริงของผลเฉลยสมบูรณ์
- จำนวนภายในวงกลมคือ ขอบเขตล่างของต้นทุนนี่คำนวณได้ที่ปม ถ้าปม x มีขอบเขตล่างของต้นทุนเป็น CLB(x) หมายความว่า ต้นทุนของปมทั้งหลายในต้นไม้ย่อยที่มี x เป็นราก จะมีต้นทุนไม่ต่ำกว่าCLB(x)

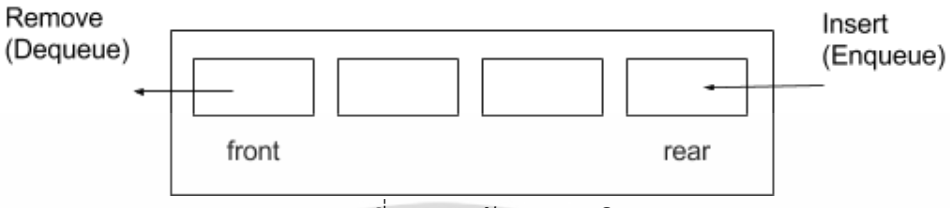
ให้ CMINเป็นต้นทุนน้อยสุดของผลเฉลยสมบูรณ์ที่ได้พบมา จากรูปสรุปว่า ค้นมาถึงตอนนี้เราได้พบผลเฉลยมาแล้ว 3 ผลเฉลย (ดูที่ปมสี่เหลี่ยม) และเท่าที่ได้ค้นมา พบผลเฉลยสมบูรณ์ที่มีต้นทุนน้อยสุดคือ 85 (CMIN= 85) แสดงว่า การค้นต่อจากนี้ไปไม่จำเป็นต้องไปค้นบริเวณที่มีต้นทุนเกิน 85 กลับไปดูที่รูปพบว่าตอนนี้เรามีปมเป็น ที่ยังไม่ได้แตกกิ่ง(ปมดำ) อยู่ 3 ปม แต่ละปมมี CLB(x) เป็น 100, 70, และ 90 สรุปว่าไม่จำเป็นต้องไปแตกกิ่งที่ปมแรกและปมที่สาม เนื่องจากมี CLB(x)> CMIN ป่วยการที่จะลงไปค้น ค้นแล้วก็ไม่มีทางได้น้อยกว่า 100 กับ 90 จึงไม่มีทางได้ดีกว่า 85 เสมือนเป็นการเล็มต้นไม้ย่อยออกได้สองต้น ค่า CLB(x) เป็นขอบเขต “ล่าง” ของต้นทุนของปม x ในขณะที่ค่า CMIN เป็นเสมือนขอบเขต “บน” ของต้นทุนในการค้น คือจะไม่ค้นในบริเวณที่มีต้นทุนเกิน CMIN เรียกว่า เป็นการจำกัดเขตหรือบริเวณของการค้นนั่นเอง

แนวคิดของการจำ CMIN และการคำนวณ CLB(x) เพื่อใช้จำกัดเขตของการค้นนี้ สามารถนำไปใช้ร่วมกับการค้นทั้งตามแนวลึก ตามแนวกว้าง และตามต้นทุนน้อยสุด แต่มักพบว่าใช้คู่กับการค้นตามต้นทุนน้อยสุด โดยใช้ CLB(x) เป็นตัวช่วยในการเลือกปมมาเป็นปมขยายเพื่อแตกกิ่ง ซึ่งก็เห็นได้ว่ามีเหตุมีผล เพราะเราก็อยากเลือกแตกกิ่ง ณ ปมซึ่งมีสิทธิ์จะได้ต้นทุนน้อยสุดก่อน ดังนั้น หากใช้การค้นตามต้นทุนน้อยสุดในรูป จะได้ลำดับของปมที่ถูกแตกกิ่งเป็นไปตามตัวเลขที่แสดงกำกับข้างปม การรู้จักเลือกปมที่จะมาเป็นปมขยายที่เหมาะสม ผนวกกับการจำกัดเขตการค้น จึงเป็นที่มาของกลวิธีขยายและจำกัดเขตที่ทำให้การค้นคำตอบในปริภูมิสถานะกระทำได้อย่างมีประสิทธิภาพ

2.1.8 คิว (Queue)

คิว [6] เป็นโครงสร้างข้อมูลแบบเชิงเส้น ซึ่งมีลักษณะเป็นแถวรอคอย ในการเพิ่มข้อมูลเข้าไปในคิว สามารถเพิ่มได้ตรง ส่วนท้ายคิวที่เรียกว่า Rear ในขณะที่การลบข้อมูลจะกระทำตรง ส่วนหัวคิวที่เรียกว่า Front

ลักษณะการทำงานของคิวเป็นลักษณะของการเข้าก่อน ออกก่อนหรือที่เรียกว่า FIFO (First In - First out) แสดงดังรูปที่ 2.23



รูปที่ 2.23 หลักการของคิว

1. การดำเนินงานของคิว (Queue Operations)

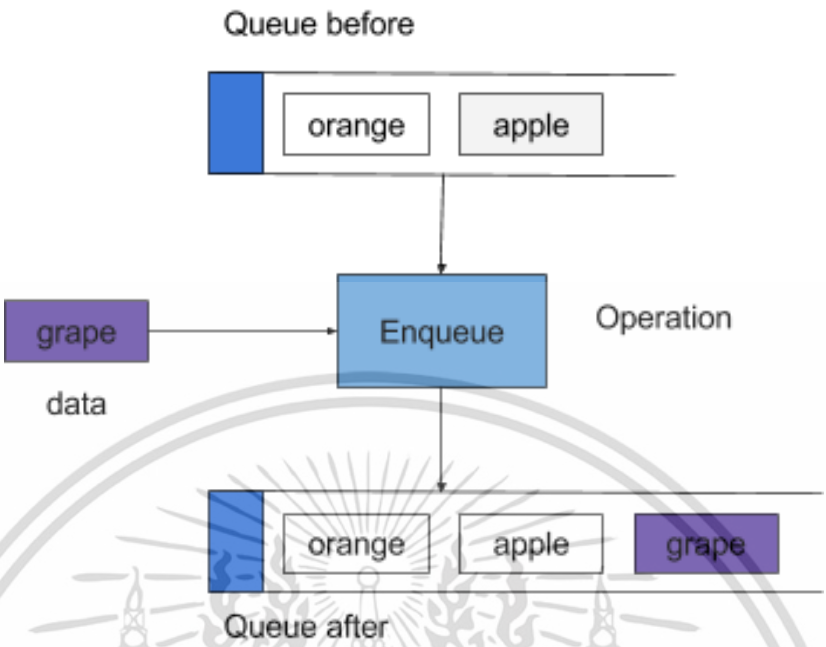
การดำเนินการพื้นฐานของคิว ข้อมูลสามารถเพิ่มได้ตรงส่วน Rear ในขณะที่การลบข้อมูลจะลบตรงส่วน Front ส่วนการอ่านข้อมูลก็สามารถทำการอ่านได้ทั้งส่วน Front และ Rear ถึงแม้ว่าโครงสร้างข้อมูลแบบคิวจะมีส่วนคล้ายคลึงกับโครงสร้างข้อมูลแบบสแต็ก แต่หนึ่งในโครงสร้างที่แสดงถึงความแตกต่างระหว่างโครงสร้างทั้งสองก็คือ ในการสร้างคิวเพื่อใช้งานจำเป็นต้องมีส่วนที่ใช้ติดตามทั้งส่วนหัวคิวและท้ายคิว นั่นหมายความว่าจะต้องมีพอยน์เตอร์ถึงสองตัว โดยพอยน์เตอร์แต่ละตัวจะชี้ตรงตำแหน่งปลายทั้งสองด้าน ซึ่งก็คือส่วน Front และ Rear นั่นเอง ในขณะที่สแต็กจะมีเพียงพอยน์เตอร์ตัวเดียวที่ใช้จัดการกับข้อมูลตรงส่วน Top ของสแต็กนั่นเอง

ฟังก์ชันการดำเนินงานพื้นฐานของคิวประกอบด้วย

1. ฟังก์ชัน Enqueue
2. ฟังก์ชัน Dequeue
3. ฟังก์ชัน Queue Front
4. ฟังก์ชัน Queue Rear

ฟังก์ชัน Enqueue

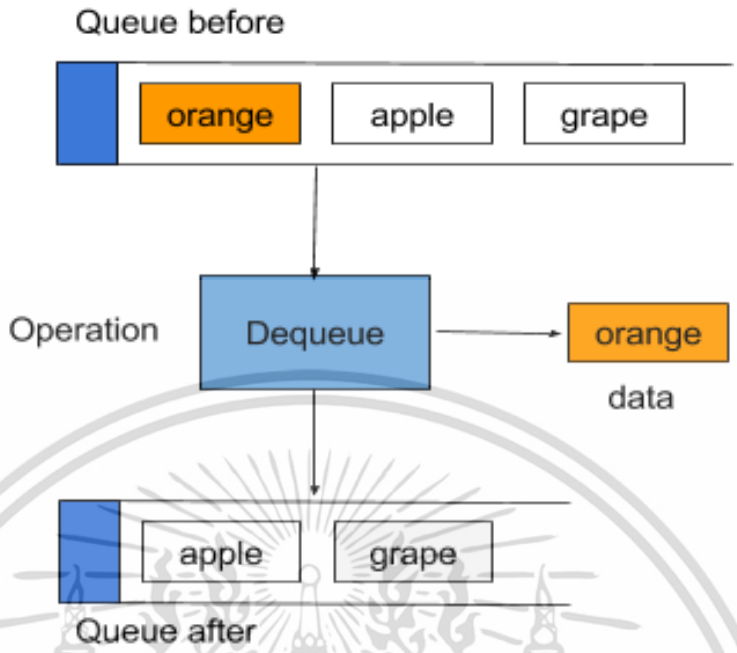
ฟังก์ชัน Enqueue เป็นการนำข้อมูลเพิ่มเข้าไปในคิว โดยหลังจากที่ข้อมูลได้เพิ่มเข้าไปในคิวแล้ว สมาชิกใหม่ที่เพิ่มเข้าไปจะนำไปต่อจากส่วน Rear ซึ่งการเพิ่มข้อมูลเข้าไปในคิว ก็ไม่ได้แตกต่างไปจากสแต็กตรงที่จะต้องมีพื้นที่ว่างพอที่จะใส่สมาชิกเข้าไป ดังนั้นหากมีพื้นที่ไม่เพียงพอต่อการเพิ่มสมาชิกใหม่เข้าไปในคิวก็จะเกิดสถานะ Overflow เช่นเดียวกัน โดยรูปที่ 2.24 แสดงถึงการดำเนินงานของฟังก์ชัน Enqueue



รูปที่ 2.24 แสดงการเพิ่มข้อมูลลงในคิวด้วยฟังก์ชัน Enqueue

ฟังก์ชัน Dequeue

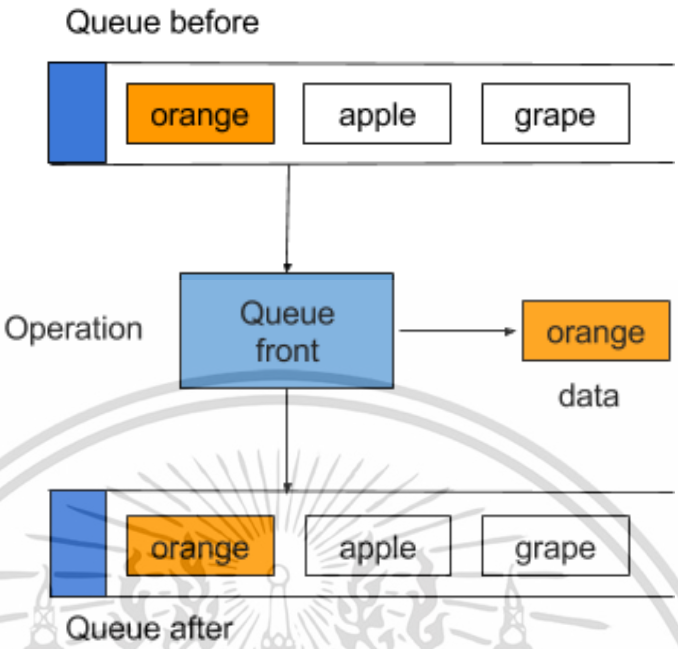
ฟังก์ชัน Dequeue เป็นการลบข้อมูลออกจากคิว โดยลบข้อมูลที่อยู่ส่วน Front จะถูกคืนค่าส่งไปยังยูสเซอร์หรือโมดูลที่เรียกใช้ จากนั้นก็จะนำข้อมูลนี้ออกจากคิวในกรณีที่มีการเรียกใช้ฟังก์ชันนี้ และหากภายในคิวไม่มีข้อมูล ก็จะทำให้เกิดสถานะ Underflow โดยรูปที่ 2.25 แสดงการลบข้อมูลออกจากคิวด้วยฟังก์ชัน Dequeue



รูปที่ 2.25 แสดงการลบข้อมูลออกจากคิวด้วยฟังก์ชัน Dequeue

ฟังก์ชัน Queue Front

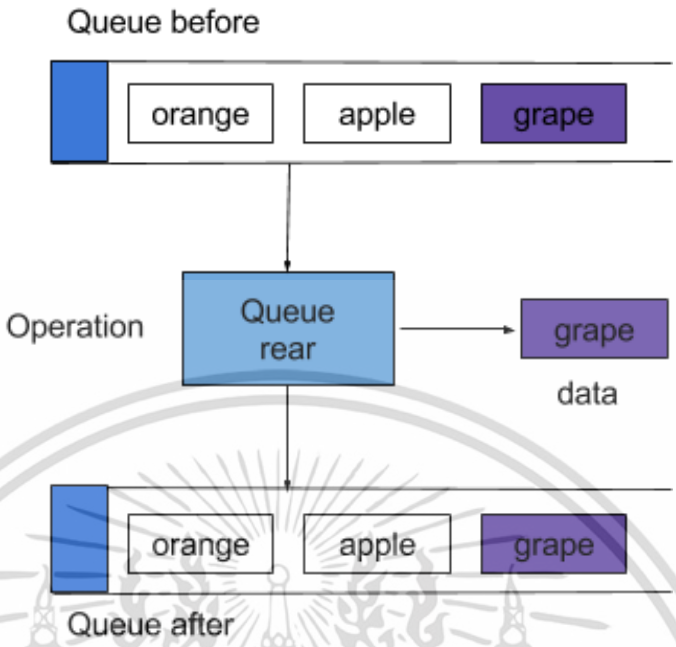
ข้อมูลที่อยู่ส่วน Front นั้น สามารถดึงขึ้นมาใช้งานได้ด้วยฟังก์ชัน Queue Front ฟังก์ชันดังกล่าวจะทำการคืนค่าข้อมูลกลับไปยังผู้เรียกใช้โดยไม่มีการเปลี่ยนแปลงใดๆในคิว อย่างไรก็ตาม หากในคิวว่างเปล่า และมีการเรียกใช้งานฟังก์ชันนี้ ก็จะทำให้เกิดสถานะข้อผิดพลาดที่เรียกว่า Underflow โดยรูปที่ 2.26 แสดงถึงการดำเนินงานของฟังก์ชัน Queue front



รูปที่ 2.26 แสดงการดึงข้อมูลส่วนหัวคิวออกมาใช้งานด้วยฟังก์ชัน Queue front

ฟังก์ชัน Queue Rear

ฟังก์ชัน Queue Rear มีลักษณะการทำงานเช่นเดียวกันกับฟังก์ชัน Queue Front แต่จะแตกต่างกันตรงที่เป็นการดึงข้อมูลส่วนท้ายคิวหรือ Rear ออกมาใช้งาน ในกรณีที่เรียกใช้ฟังก์ชันนี้และปราฏกว่าภายในคิวว่างเปล่า ก็จะทำให้เกิดสถานะข้อผิดพลาดที่เรียกว่า Underflow โดยรูปที่ 2.27 แสดงถึงการดำเนินงานของฟังก์ชัน Queue Rear



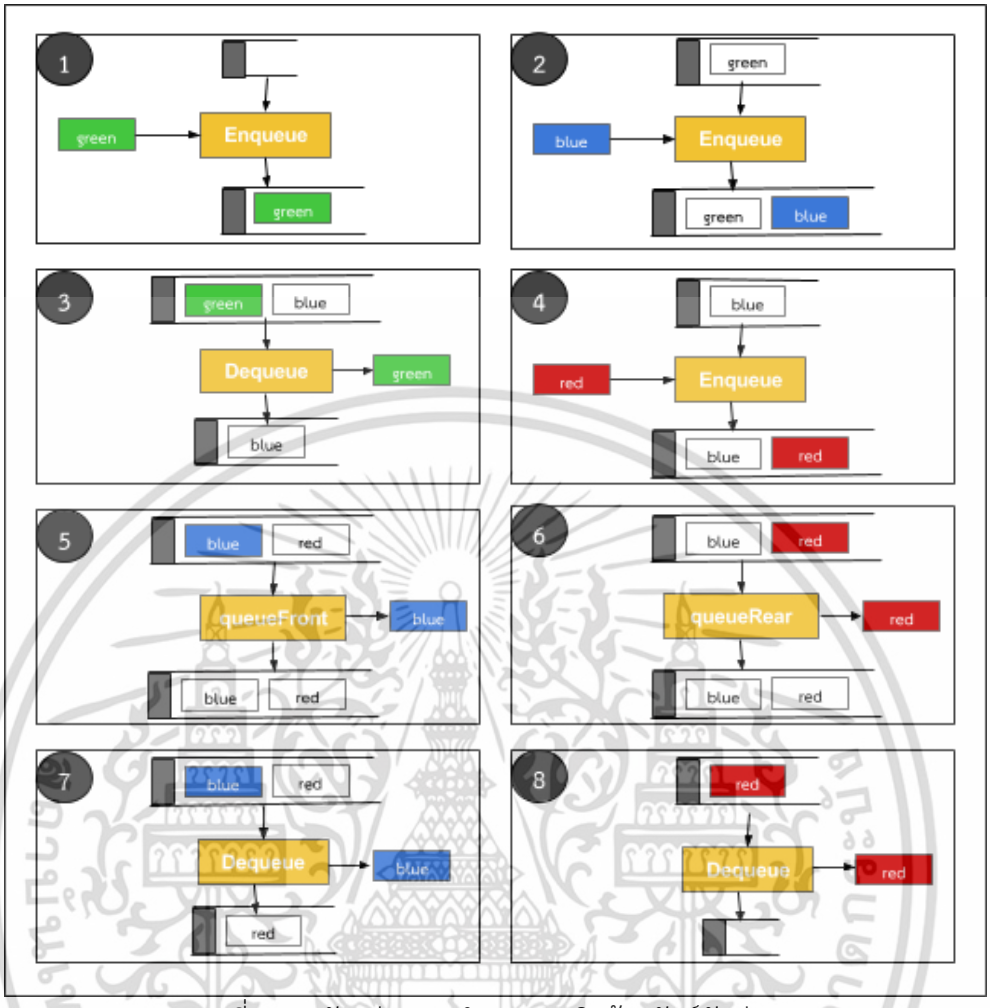
รูปที่ 2.27 แสดงการดึงข้อมูลส่วนท้ายคิวออกมาใช้งานด้วยฟังก์ชัน Queue rear

2. ตัวอย่างการทำงานของคิว (Queue Example)

จากรูปที่ 2.28 ต่อไปนี้ เป็นตัวอย่างการดำเนินงานพื้นฐานของคิว ซึ่งประกอบไปด้วยฟังก์ชันทั้ง 4 ที่ได้กล่าวไว้ข้างต้น โดยมีการดำเนินงานต่อไปนี้

1. เริ่มต้นจากคิวว่าง และได้มีการใส่ข้อมูลสมาชิกตัวแรกเข้าไปในคิว คือ green
2. ได้ใสข้อมูลสมาชิก blue ลงไปในคิว ทำให้ ณ ขณะนี้ คิวมีข้อมูลอยู่สองสมาชิกด้วยกัน
3. มีการนำข้อมูลตรงส่วน Front ออกไป จึงทำให้ภายในคิวเหลือสมาชิกเดียว คือ blue
4. หลังจากนั้น มีการใส่ข้อมูลสมาชิก red เข้าไปในคิวเพิ่มเติม
5. ดึงข้อมูลสมาชิกส่วน Front ซึ่งอยู่หัวคิวออกมาใช้งาน ข้อมูลที่ได้คือสมาชิก blue
6. ได้มีการดึงข้อมูลส่วน Rear ซึ่งอยู่ท้ายคิวออกมาใช้งาน ข้อมูลที่ได้คือสมาชิก red
7. ทำการลบข้อมูลออกจากคิว ทำให้ภายในคิวเหลือเพียงสมาชิก red
8. นำข้อมูลสมาชิก red ออกจากคิว ส่งผลให้คิวว่างเปล่า

ตัวอย่าง การทำงานของคิวด้วยฟังก์ชันต่างๆ แสดงดังรูปที่ 2.28



รูปที่ 2.28 ตัวอย่างการทำงานของคิวด้วยฟังก์ชันต่างๆ

2.1.9 การวิเคราะห์อัลกอริทึม (Algorithm Analysis)

1. ความรู้พื้นฐานเกี่ยวกับโครงสร้างข้อมูลและอัลกอริธึม

1.1 นิยามโครงสร้างข้อมูลและอัลกอริธึม

โครงสร้างข้อมูล

โครงสร้างข้อมูล (Data Structure) หมายถึง การรวมประเภทข้อมูล (Data Type) เข้าไว้ด้วยกัน จนกระทั่งกลายเป็นกลุ่มประเภทข้อมูล และมีการกำหนดคำนิยามของความสัมพันธ์ภายในกลุ่มข้อมูลไว้อย่างชัดเจน ซึ่งการรวมกลุ่มฐานอาจเป็นการรวมกลุ่มกันระหว่างข้อมูล ประเภทเดียวกัน ต่างประเภทกัน หรือต่างโครงสร้างข้อมูลกันก็ได้

ส่วนความสัมพันธ์ภายในกลุ่มข้อมูลนั้น คือ กลุ่มของกฎระเบียบข้อบังคับต่างๆ ที่ใช้ผูกข้อมูลเข้าไว้ด้วยกันอย่างเป็นระบบ นอกจากนั้นเรายังสามารถสร้างโครงสร้างข้อมูลขึ้นมาใหม่ ซึ่งภายในประกอบ

ไปด้วยโครงสร้างข้อมูลอื่นๆ มารวมกันก็ได้เพื่อใช้งานร่วมกับโปรแกรมที่เราเขียน ซึ่งในปัจจุบันภาษาคอมพิวเตอร์ส่วนใหญ่ก็สามารถรองรับการทำงานของโครงสร้างข้อมูลได้หลายแบบ

อัลกอริธึม

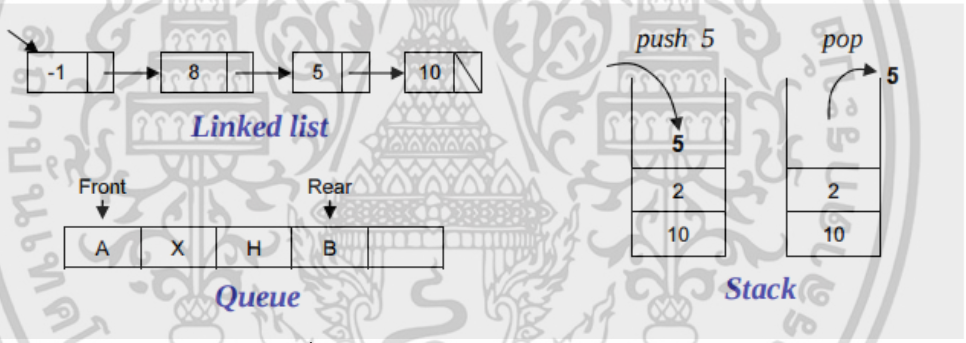
อัลกอริธึม (Algorithm) หมายถึง ลำดับขั้นตอนวิธีในการทำงานของโปรแกรมเพื่อแก้ปัญหาใดปัญหาหนึ่ง ซึ่งถ้าปฏิบัติตามขั้นตอนอย่างถูกต้องแล้วจะต้องสามารถช่วยแก้ปัญหาหรือประมวลผลตามต้องการได้สำเร็จ

1.2 ประเภทของโครงสร้างข้อมูล

ตามปกติแล้ว โครงสร้างข้อมูลสามารถแบ่งออกเป็น 2 ประเภทใหญ่ ๆ ด้วยกัน คือ

1) โครงสร้างข้อมูลแบบเชิงเส้น (Linear Data Structure)

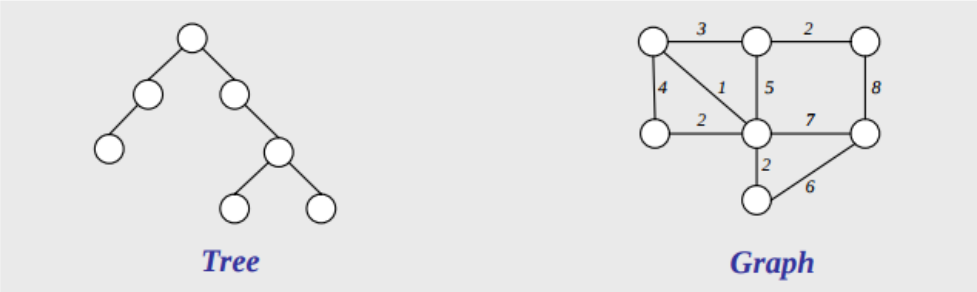
โครงสร้างข้อมูลชนิดนี้มีรูปแบบเป็นรายการต่อเนื่อง ข้อมูลที่จัดเก็บจะมีลักษณะเป็นแถวลำดับต่อเนื่องกันไป ซึ่งเป็นไปในลักษณะแนวเส้นตรงนั่นเอง ตัวอย่าง โครงสร้างข้อมูลแบบเชิงเส้นประกอบด้วย Array, Linked list, Stack และ Queue แสดงได้ดังรูปที่ 2.29



รูปที่ 2.29 โครงสร้างข้อมูลแบบเชิงเส้น

2) โครงสร้างข้อมูลแบบไม่เป็นเชิงเส้น (Non-Linear Data Structure)

โครงสร้างข้อมูลชนิดนี้จะตรงกันข้ามกับแบบแรก นั่นคือจะไม่เป็นลักษณะแนวเส้นตรง โดยตัวอย่างโครงสร้างข้อมูลแบบไม่เป็นเชิงเส้น เช่น Tree และ Graph เป็นต้น แสดงได้ดังรูปที่ 2.30



รูปที่ 2.30 โครงสร้างข้อมูลแบบไม่เป็นเชิงเส้น

1.3 รหัส Pseudo (Pseudo Code)

รหัส Pseudo ก้ำกึ่งระหว่างภาษาอังกฤษกับภาษาคอมพิวเตอร์ใช้ในการอธิบายลักษณะโครงสร้างข้อมูลและการทำงานของอัลกอริธึมที่เราเขียนขึ้นทำให้ไม่ต้องเขียนอธิบายด้วย code ของภาษาคอมพิวเตอร์ใดภาษาหนึ่ง มีความยืดหยุ่นตามหลักไวยากรณ์ของภาษาคอมพิวเตอร์แทบทุกภาษา นอกจากนั้นยังช่วยให้นักเขียนโปรแกรมและผู้ที่ไม่เคยเขียนโปรแกรมมาก่อนสามารถเข้าใจโครงสร้างของข้อมูลและการทำงานของอัลกอริธึมที่เราเขียนขึ้น ได้โดยง่าย

ตัวอย่าง Pseudo Code 1 แสดงได้ดังรูปที่ 2.31

```
1. get number of quizzes
2. sum = 0
3. count = 0
4. while count < number of quizzes
    4.1  get quiz grade
    4.2  add quiz grade to sum; i.e. sum = sum + quiz grade
    4.3  add 1 to count; i.e. count = count + 1
5. average = sum divided by number of quizzes
6. display average
7. stop
```

รูปที่ 2.31 ตัวอย่าง Pseudo Code 1

ตัวอย่าง Pseudo Code 2 แสดงได้ดังรูปที่ 2.32

```
1. get hours worked
2. get pay rate
3. if hours worked <=  40 then
     3.1  gross pay = pay rate times hours worked
4. else
     4.1  gross pay = pay rate times 40 plus 1.5 times pay rate times
5. display gross pay
6. stop
```

รูปที่ 2.32 ตัวอย่าง Pseudo Code 2

2. การวิเคราะห์ประสิทธิภาพของอัลกอริธึม

2.1 ประสิทธิภาพของอัลกอริธึม

ในการเขียนโปรแกรมคอมพิวเตอร์สิ่งที่สำคัญที่สุดที่จะต้องคำนึงถึง คือ ผลลัพธ์ของการประมวลผลโดยจะต้องมีความถูกต้อง แม่นยำ สามารถแก้ปัญหาได้ตรงตามที่ตั้งใจไว้

ลักษณะสำคัญของโปรแกรมคอมพิวเตอร์ที่ดีไม่ควรใช้เวลาในการประมวลผลนานจนเกินไป ซึ่งการที่จะทำให้โปรแกรมคอมพิวเตอร์ลดเวลาในการประมวลผลลงได้นั้น จะต้องออกแบบอัลกอริธึมให้มีประสิทธิภาพ

ประสิทธิภาพของอัลกอริธึม [11] จะพิจารณาอยู่ 2 ส่วนหลัก ๆ ได้แก่

- หน่วยความจำ (Memory) ที่จะต้องใช้ในการประมวลผล
- เวลา (Time) ที่จะต้องใช้ในการประมวลผล

โดยทั้ง 2 สิ่งนี้มักถูกใช้เป็นตัวตัดสินประสิทธิภาพของอัลกอริธึมว่ามีประสิทธิภาพมากหรือน้อยเพียงใด ดังนั้น ถ้าจำเป็นจะต้องดำเนินการแก้ไขปรับปรุงเพื่อเพิ่มประสิทธิภาพของอัลกอริธึมจำเป็นจะต้องพิจารณาก่อนว่าจะสามารถดำเนินการแก้ไขกับส่วนใดได้บ้างและส่งผลกระทบต่อกันหรือไม่อย่างไร

การประเมินประสิทธิภาพของอัลกอริธึม

การประเมินประสิทธิภาพของอัลกอริธึมแบ่งออกเป็น 2 วิธีคือ การวิเคราะห์และวัดผล ดังนี้

- การวิเคราะห์ประสิทธิภาพของอัลกอริธึม (Performance Analysis) จะใช้วิธีการวิเคราะห์วิธีการทำงานของอัลกอริธึม
- การวัดประสิทธิภาพของอัลกอริธึม (Performance Measurement) เป็นการวัดผลจากการทดลองจริง

การวิเคราะห์ประสิทธิภาพของอัลกอริธึม

การวิเคราะห์ประสิทธิภาพของอัลกอริธึมแบ่งออกเป็น 2 ส่วน ดังนี้

- การวิเคราะห์หน่วยความจำที่ต้องใช้ในการประมวลผล (Space Complexity)
- การวิเคราะห์เวลาที่ต้องใช้ในการประมวลผล (Time Complexity)

2.2 การวิเคราะห์ Space Complexity

การวิเคราะห์ Space Complexity ของอัลกอริธึม คือ การวิเคราะห์ว่าจะต้องใช้หน่วยความจำทั้งหมดเท่าไรในการประมวลผลอัลกอริธึมนั้น สาเหตุที่ต้องทราบจำนวนของหน่วยความจำที่จะต้องใช้นั้นมีเหตุผลดังนี้

- ทำให้เราทราบว่าอัลกอริธึมนั้นสามารถรองรับจำนวนข้อมูลที่ส่งเข้ามาประมวลผล (Input Data) ได้มากที่สุดเท่าใด เพื่อให้อัลกอริธึมนั้นสามารถประมวลผลได้อยู่

- กรณีที่ต้องประมวลผลบนเครื่องคอมพิวเตอร์ที่ใช้งานรวมกันหลายคนในเครือข่ายจำเป็นจะต้องทราบขนาดของหน่วยความจำที่จะต้องใช้ในการประมวลผลอัลกอริธึม เพื่อไม่ให้กระทบกับการทำงานของคนอื่น
- เพื่อเลือกคุณลักษณะของคอมพิวเตอร์ที่จะใช้ติดตั้งโปรแกรมที่พัฒนาขึ้นได้อย่างเหมาะสม เพื่อถ้านำไปติดตั้งที่เครื่องที่มีหน่วยความจำไม่เพียงพอโปรแกรมก็จะไม่ทำงาน

องค์ประกอบของ Space Complexity

การวิเคราะห์ Space Complexity ประกอบด้วย 3 ส่วน คือ Instruction Space, Data Space และ Environment Stack Space ดังนี้

1) Instruction Space

คือ จำนวนของหน่วยความจำที่คอมไพเลอร์จำเป็นต้องใช้ขณะทำการคอมไพล์โปรแกรม ซึ่งจำนวนหน่วยความจำที่ต้องใช้จะขึ้นอยู่กับคอมไพเลอร์แต่ละประเภท

2) Data Space

คือ จำนวนหน่วยความจำที่ต้องใช้สำหรับเก็บค่าคงที่และตัวแปรทั้งหมดที่ต้องใช้ในการประมวลผลโปรแกรม ซึ่ง Data Space แบ่งออกเป็น 2 ประเภท คือ

- หน่วยความจำแบบ static คือ จำนวนของหน่วยความจำที่ต้องใช้อย่างแน่นอนไม่มีการเปลี่ยนแปลง ประกอบด้วยหน่วยความจำที่ใช้เก็บค่าคงที่และตัวแปรประเภท array
- หน่วยความจำแบบ dynamic คือ จำนวนของหน่วยความจำที่ใช้ในการประมวลผลสามารถเปลี่ยนแปลงได้และจะทราบจำนวนหน่วยความจำที่จะใช้ก็ต่อเมื่อโปรแกรมกำลังทำงานอยู่

3) Environment Stack Space

คือ หน่วยความจำที่ต้องใช้ในการเก็บผลลัพธ์ของข้อมูลเอาไว้เพื่อรอเวลาที่จะนำผลลัพธ์นั้นกลับไปประมวลผลอีกครั้ง หน่วยความจำประเภทนี้จะเกิดขึ้นเมื่อมีการร้องขอให้นำมาใช้เท่านั้น

เทคนิคการเขียนโปรแกรมคอมพิวเตอร์ที่ต้องร้องขอพื้นที่ในหน่วยความจำประเภทนี้ มาใช้ก็คือการทำ Recursive โดยหน่วยความจำที่ต้องใช้ก็จะขึ้นอยู่กับความลึกของการทำ Recursive ด้วย ยิ่งลึกมากก็จะยิ่งใช้หน่วยความจำมากขึ้นตามไปด้วย

<u>ตัวอย่าง</u> การวิเคราะห์ Space Complexity แสดงได้ดังรูปที่ 2.33

```
{
        int num1, num2, temp;
        temp = num1;
        num1 = num2;
        num2 = temp;
}
```

รูปที่ 2.33 ตัวอย่างการวิเคราะห์ Space Complexity

จากตัวอย่าง ตัวแปรที่ใช้จะมีอยู่ด้วยกัน 3 ตัวแปร ประกอบด้วย num1, num2 และ temp ซึ่งเป็นตัวแปรชนิด integer ทั้งหมดโดยจะใช้หน่วยความจำในการเก็บตัวแปรประเภท integer ตัวแปรละ 2 bytes ดังนั้น จะต้องใช้หน่วยความจำทั้งสิ้น 3x2 = 6 bytes

ตัวอย่าง การทำงานแบบ recursive แสดงได้ดังรูปที่ 2.34

```
int factorial(int n)
{
        if (n==0)
                return 1;
        else
                return (n * factorial(n-1));
}
```

รูปที่ 2.34 ตัวอย่างการทำงานแบบ recursive

ในตัวอย่างนี้มีตัวแปรที่ใช้ 1 ตัวคือ n ซึ่งเป็นตัวแปรชนิด integer แต่ลักษณะการ ทำงานเป็นแบบ recursive ดังนั้น หน่วยความจำส่วนใหญ่จะไปขึ้นอยู่กับความลึกของการทำ recursive ซึ่งจะเห็นว่าการประมวลผลของคำสั่ง if จะมีอยู่ด้วยกัน 2 แบบ คือ เมื่อ n = 0 และ เมื่อ n มีค่าใดๆ ดังนั้น ความลึกของการทำ recursive จะมีค่าเท่ากับ 1 (กรณีเมื่อ n = 0) หรือมี ค่าเท่ากับ n (เมื่อ n เป็นค่าใดๆ) เท่านั้น

สรุปออกมาได้ว่า ความลึกของการทำ Recursive คือ ค่าที่มากที่สุดระหว่าง 1 กับ n สามารถแทนได้ด้วยสัญลักษณ์ Max {1, n}

ขั้นต่อไปจะพิจารณาว่า แต่ละครั้งที่มีการเรียกใช้ฟังก์ชัน Factorial จะต้องใช้ หน่วยความจำในการเก็บข้อมูลเท่าไร ซึ่งจากตัวอย่างจะต้องใช้หน่วยความจำทั้งสิ้น 4 bytes (สำหรับเก็บ address 2 bytes และตัวแปรชนิด integer อีก 2 bytes) ทำให้สามารถสรุปได้ว่า ต้องใช้หน่วยความจำสำหรับอัลกอริธึมที่ 2 ทั้งหมดเท่ากับ 4xMax{1, n} bytes

2.3 การวิเคราะห์ Time Complexity

Time Complexity คือ เวลาที่เครื่องคอมพิวเตอร์ต้องใช้ในการประมวลผลอัลกอริธึม เหตุผลที่ควรทราบเวลาที่ต้องใช้มีหลายประการด้วยกัน ยกตัวอย่างเช่น

- ทำให้สามารถประมาณเวลาทั้งหมดที่ต้องใช้ในโปรแกรม
- สามารถมุ่งประเด็นการแก้ไขไปที่อัลกอริธึมที่ใช้เวลาในการประมวลผลนานๆ ทำให้ไม่ต้องแก้ไขทั้งโปรแกรม

- โปรแกรมคอมพิวเตอร์ที่ทำงานแบบ Interactive เวลาที่ใช้ในการประมวลผลแต่ละขั้นตอนจะเป็นสิ่งสำคัญ เพราะผู้ใช้ไม่ควรรอการประมวลผลเป็นเวลานานในการ Interactive แต่ละครั้ง
- สามารถเลือกคุณลักษณะของคอมพิวเตอร์ที่จะใช้ติดตั้งโปรแกรมที่พัฒนาขึ้นได้อย่างเหมาะสม

หลักในการพิจารณาอย่างคร่าวๆ ถึงเวลาที่ต้องใช้ในการประมวลผล เช่น

- โปรแกรมจะสามารถประมวลผลได้เร็วกว่า เมื่ออยู่บนเครื่องคอมพิวเตอร์ที่มีความเร็วในการประมวลผลสูงกว่า
- ถ้าใช้คอมไพเลอร์ตัวเดียวกัน code ที่สั้นกว่าย่อมใช้เวลาในการประมวลผลได้น้อยกว่าเวลาในการประมวลผลของโปรแกรม
- Compile Time คือ เวลาที่ใช้ในการตรวจสอบไวยากรณ์ (syntax) ของ code ว่าเขียนได้ถูกต้องหรือไม่ถ้ามีข้อผิดพลาดเกิดขึ้นคอมไพเลอร์จะแจ้งเตือนให้ทราบ
- Run Time หรือ Execution Time คือ เวลาที่เครื่องคอมพิวเตอร์ใช้ในการประมวลผล

การนับตัวดำเนินการ

ในการวิเคราะห์ Time Complexity วิธีที่มักใช้กันบ่อยๆ คือ การนับตัวดำเนินการ (Operation count) ในอัลกอริธึม โดยพิจารณาลักษณะของตัวดำเนินการควบคู่ไปด้วยสามารถสรุปออกมาได้เป็นหลักคร่าวๆ ดังนี้

1) แบบ Linear Loops

อัลกอริธึมมีการทำงานแบบวนรอบ (Loop) โดยแต่ละ loop จะมีการเพิ่มหรือลดค่าในปริมาณที่คงที่ เช่น เพิ่มค่าตัวแปรขึ้นทีละหนึ่งในแต่ละรอบ แสดงได้ดังรูปที่ 2.35

```
x = 1
Loop (x <= 2000)
    x = x+5
```

```
x = 1
Loop (x <= 2000)
    x = x+1
```

รูปที่ 2.35 การทำงานแบบ Linear Loops

ถ้าให้ f(n) แทนประสิทธิภาพ และ n แทนจำนวนรอบการทำงานสามารถเขียนเป็นสมการวัดประสิทธิภาพของอัลกอริธึมแบบ Linear loop ได้ดังนี้

$$f(n) = n$$

2) แบบ Logarithmic Loops

อัลกอริธึมจะทำงานแบบ Loop โดยการทำงานภายในแต่ละ loop จะเพิ่มหรือลดค่าเป็นเท่าตัว เช่น คูณเพิ่มค่าตัวแปรขึ้นทีละ 2 เท่าในแต่ละรอบ แสดงได้ดังรูปที่ 2.36

```
x = 1000
Loop (x >= 1)
    x = x/2

x = 1
Loop (x < 1000)
    x = x*2
```

รูปที่ 2.36 การทำงานแบบ Logarithmic Loops

ถ้าให้ f(n) แทนประสิทธิภาพ และ n แทนจำนวนรอบการทำงาน สามารถเขียนเป็นสมการวัดประสิทธิภาพของอัลกอริธึมแบบ Linear loop ได้ดังนี้

$$f(n) = [\log n]$$

3) แบบ Nested Loops

คือ ลักษณะของอัลกอริธึมที่มี loop ซ่อนอยู่ภายใน loop โดยประสิทธิภาพของอัลกอริธึมก็จะมีค่าเท่ากับจำนวน loop ทั้งหมดที่จะต้องประมวลผล ซึ่งหาได้จากการเอาจำนวน loop ที่ซ่อนกันมาคูณกัน

2.4 สภาวะประสิทธิภาพของอัลกอริธึม

ลักษณะของกลุ่มข้อมูลที่เข้ามาประมวลผล สามารถที่จะแบ่งประสิทธิภาพของอัลกอริธึมออกเป็น 3 สภาวะด้วยกัน ซึ่งในการวิเคราะห์ประสิทธิภาพของอัลกอริธึมเราจะต้องมีการระบุด้วยว่า ค่าผลลัพธ์ที่เราได้จากการวิเคราะห์นั้น พิจารณาเมื่ออัลกอริธึมอยู่ในสภาวะใด ซึ่งแบ่งออกเป็น

- Best-case

คือ การวิเคราะห์อัลกอริธึมเมื่อข้อมูลที่เข้ามาประมวลผลส่งผลให้อัลกอริธึมมีประสิทธิภาพดีที่สุดและใช้เวลาในการประมวลผลน้อยที่สุดด้วย

- Worst-case

คือ การวิเคราะห์อัลกอริธึมเมื่อข้อมูลที่เข้ามาประมวลผลส่งผลให้อัลกอริธึมมีประสิทธิภาพแย่ที่สุดและใช้เวลาในการประมวลผลนานที่สุดด้วย ซึ่งกรณีนี้ค่าที่ได้จากการวิเคราะห์ต้องมีค่ามากที่สุดเมื่อเทียบกับกรณีอื่น

- Average-case

คือ การวิเคราะห์อัลกอริธึมเมื่อข้อมูลที่เข้ามาประมวลผลส่งผลให้อัลกอริธึมโดยเฉลี่ยมีค่าประมาณที่สามารถระบุได้

โดยส่วนใหญ่มักนิยมเปรียบเทียบประสิทธิภาพของอัลกอริธึมในสภาวะ Worst-case เนื่องจากเป็นสภาวะที่ใช้หน่วยความจำมากที่สุดและใช้เวลาในการประมวลผลนานที่สุด จึงถือ เป็นอีกสิ่งหนึ่งที่ใช้รับประกันได้ว่า อัลกอริธึมนั้นๆ จะไม่ทำงานแย่ไปกว่าค่านี้อีกแล้วใช้ในการรับประกันประสิทธิภาพของอัลกอริธึม

2.5 สัญลักษณ์ Asymptotic

สัญลักษณ์ Asymptotic คือ ฟังก์ชันของเวลาที่ใช้ในการประมวลผลอัลกอริธึมนั้น ๆ โดยพิจารณาค่าเมื่ออัลกอริธึมนั้นมีปริมาณข้อมูลมาก ๆ สมมติให้มีค่าเป็นอนันต์ (Infinity) แล้วพิจารณาว่าต้องใช้เวลาในการประมวลผลมีขอบเขตของแนวโนมการเติบโตทางเวลา (Growth in Run Time) เป็นอย่างไร ซึ่งเขียนแทนออกมาเป็นฟังก์ชันทางคณิตศาสตร์หรืออาจกล่าวได้ว่า เพื่อเป็นการอธิบายให้เห็นภาพรวมของพฤติกรรมทางเวลาของอัลกอริธึมนั้นๆ

ฟังก์ชันการเติบโตทางเวลา (Growth in Run Time)

ค่าฟังก์ชันที่ใช้อธิบายพฤติกรรมแนวโน้มการเติบโตทางเวลาของอัลกอริธึมหรือฟังก์ชันเติบโตทางเวลา (Growth in Run Time) นั้นเป็นค่าฟังก์ชันทางคณิตศาสตร์ที่แสดงถึงความสัมพันธ์ระหว่างปริมาณข้อมูลที่กำลังประมวลผลกับเวลาที่ต้องใช้ในการประมวลผลข้อมูลนั้นให้เสร็จว่ามีความสัมพันธ์กันอย่างไร การคำนวณหาค่า Run Time ของอัลกอริธึมหนึ่ง ๆ เราสามารถหาได้จากการคำนวณ Step Counts

การคำนวณ Step Counts จะพิจารณาหางานทั้งหมดที่อัลกอริธึมนั้นต้องทำนั่นคือ หาว่าจะต้องประมวลผลคำสั่งทั้งหมดที่อยู่ภายในอัลกอริธึมเป็นจำนวนรวมทั้งสิ้นกี่ครั้ง (กี่ step)

ตัวอย่าง การคำนวณค่า Step Counts แสดงดังรูปที่ 2.37

code	จำนวนครั้ง(step)ที่ประมวลผล

void main()
{ int i, n;
scanf("%d",&n);
for(i=1;i<n;i++)
{ printf("hello");
printf("\n");
}
}
	0
	1
	1
	1+n+n
	n
	n

รวมทั้งสิ้น (ค่า step counts)	4n+3

รูปที่ 2.37 ตัวอย่างการคำนวณค่า

สัญลักษณ์ Big-Oh หรือ O ( )

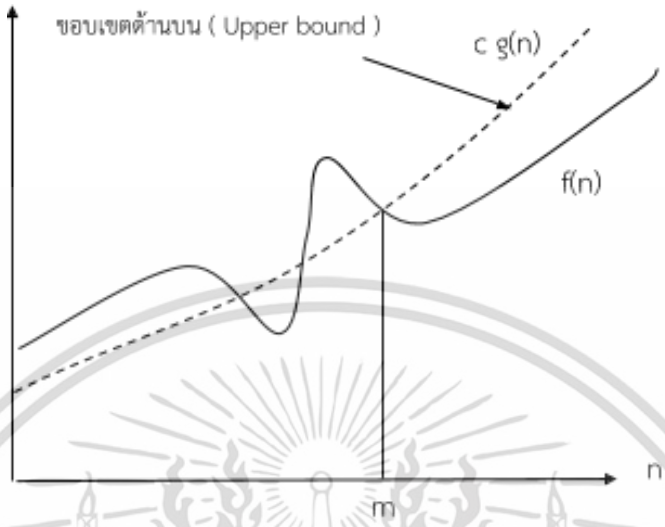
อัตราการเติบโต Big-O เป็นการวัดประสิทธิภาพเชิงเวลาที่ใช้ในการประมวลผลของอัลกอริทึมซึ่งเป็นฟังก์ชันเวลาขอบเขตบนที่ใช้ในการประมวลผล (Asymptotic upper bounds) โดยมีสัญลักษณ์เป็นตัวโอใหญ่ (O)

ดังนั้น หากใช้สัญลักษณ์ Big-O จะเป็นการรับรองว่าเวลาที่ใช้ในการประมวลผลสูงสุดของอัลกอริทึมนี้จะไม่มากกว่าเวลาที่คำนวณได้จากฟังก์ชัน Big-O เช่น O(n) หมายถึง ฟังก์ชันนี้จะใช้เวลาในการประมวลผลน้อยกว่าหรือเท่ากับ n (≤n) เสมอ

นิยาม Big-O

ฟังก์ชัน $f(n) = O(g(n))$ ก็ต่อเมื่อมีค่าคงที่ m, c ที่ทำให้ $f(n) \leq cg(n)$ เมื่อ $n \geq m$

กราฟแสดงอัตราการเติบโต Big-O แสดงดังรูปที่ 2.38



รูปที่ 2.38 กราฟแสดงอัตราการเติบโต Big-O

จากนิยามสามารถอธิบายได้ว่า ฟังก์ชัน f จะมีอัตราการเติบโตไม่มากไปกว่าฟังก์ชัน g เมื่อค่า n มีค่ามากกว่าหรือเท่ากับ m ดังรูปที่ 2.38 กราฟที่แสดงให้เห็นว่าฟังก์ชัน f(n) ในตำแหน่งที่ m จะเป็นตำแหน่งที่ค่าเวลาในการประมวลผลไม่มากกว่าฟังก์ชัน cg(n) เสมอ

สัญลักษณ์ Big-Omega หรือ Ω ( )

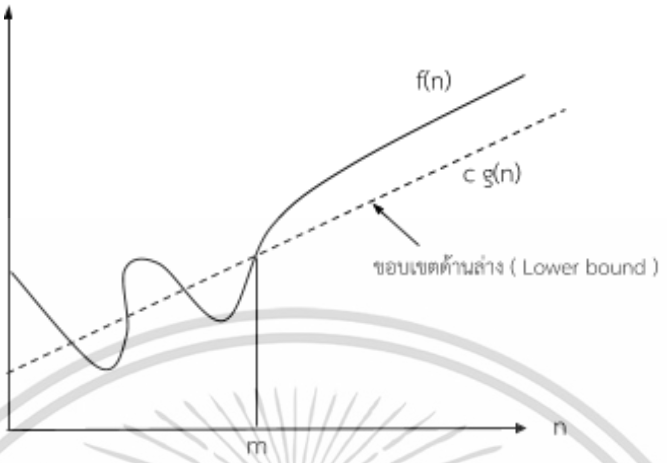
อัตราการเติบโต Big-Omega เป็นการวัดประสิทธิภาพเชิงเวลาที่ใช้ในการประมวลผลของอัลกอริทึมซึ่งเป็นฟังก์ชันเวลาขอบเขตบนที่ใช้ในการประมวลผล (Asymptotic lower bounds) โดยมีสัญลักษณ์เป็นตัวโอเมก้าใหญ่ (**Ω**)

ดังนั้น หากใช้สัญลักษณ์ Big-**Ω** จะเป็นการรับรองว่าเวลาที่ใช้ในการประมวลผลสูงสุดของอัลกอริทึมนี้จะไม่น้อยกว่าเวลาที่คำนวณได้จากฟังก์ชัน Big-**Ω** เช่น **Ω**(n) หมายถึง ฟังก์ชันนี้จะใช้เวลาในการประมวลผลมากกว่าหรือเท่ากับ n (≥n) เสมอ

<u>นิยาม Big-Ω</u>

ฟังก์ชัน f(n) = **Ω**(g(n)) ก็ต่อเมื่อมีค่าคงที่ m, c ที่ทำให้ f(n) ≥ cg(n) เมื่อ n > m

กราฟแสดงอัตราการเติบโต Big-Omega แสดงดังรูปที่ 2.39



รูปที่ 2.39 กราฟแสดงอัตราการเติบโต Big-Omega

จากนิยามสามารถอธิบายได้ว่า ฟังก์ชัน f จะมีอัตราการเติบโตเร็วกว่าฟังก์ชัน g เมื่อค่า n มีค่ามากกว่าหรือเท่ากับ m ดังรูปที่ 2.39 กราฟที่แสดงให้เห็นได้ว่าฟังก์ชัน f(n) ในตำแหน่งที่ m จะเป็นตำแหน่งที่ค่าเวลาในการประมวลผลไม่น้อยกว่าฟังก์ชัน cg(n) เสมอ

สัญลักษณ์ Big-Theta หรือ Θ ( )

อัตราการเติบโต Big-Theta เป็นการวัดประสิทธิภาพเชิงเวลาที่ใช้ในการประมวลผลของอัลกอริทึมซึ่งเป็นฟังก์ชันเวลาขอบเขตบนและขอบเขตล่างที่ใช้ในการประมวลผล โดยมีสัญลักษณ์เป็นตัวเทต้าใหญ่ (**Θ**)

ดังนั้น หากใช้สัญลักษณ์ Big-**Θ** จะเป็นการรับรองว่าเวลาที่ใช้ในการประมวลผลสูงสุดของอัลกอริทึมนี้จะอยู่ระหว่างเวลาที่คำนวณได้จากฟังก์ชัน Big-**Ω** และ Big-O เช่น **Θ**(n) หมายถึง ฟังก์ชันนี้จะใช้เวลาในการประมวลผลระหว่าง **Ω**(n) และ O(n) เสมอ

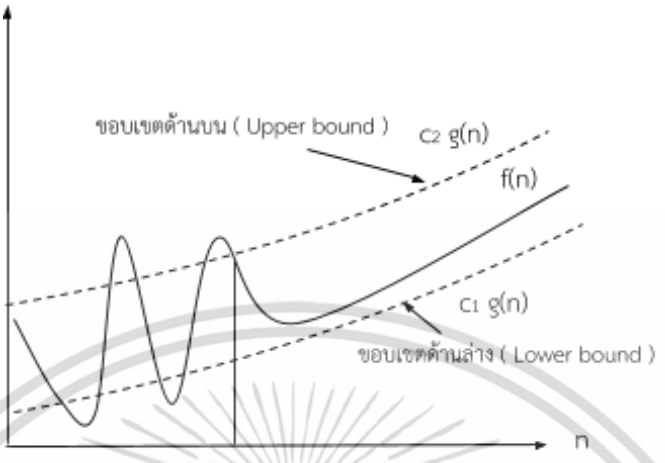
<u>นิยาม Big-**Θ**</u>

ฟังก์ชัน f(n) = **Θ**(g(n)) ก็ต่อเมื่อมีค่าคงที่ c_1, c_2 และ m ที่ทำให้ $c_1g(n) \le f(n) \le c_2g(n)$ เมื่อ $n \ge m$

โดยฟังก์ชัน $c_1g(n)$ คือ ขอบเขตด้านล่าง **Ω**(g(n))

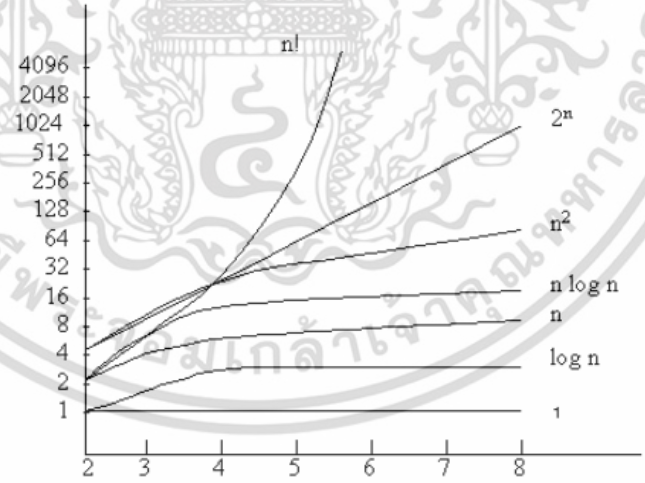
$c_2g(n)$ คือ ขอบเขตด้านบน O(g(n))

กราฟแสดงอัตราการเติบโต Big-Theta แสดงดังรูปที่ 2.40



รูปที่ 2.40 กราฟแสดงอัตราการเติบโต Big-Theta

จากนิยามสามารถอธิบายได้ว่า ฟังก์ชัน f(n) จะมีอัตราการเติบโตไม่น้อยไปกว่าฟังก์ชัน $c_1g(n)$ และไม่มากไปกว่าฟังก์ชัน $c_2g(n)$ เมื่อค่า n มีค่ามากกว่าหรือเท่ากับ m ($O(g(n)) \leq f(n) \leq (\Omega(g(n)))$ แสดงดังรูปที่ 2.41



รูปที่ 2.41 แสดงการเติบโตของฟังก์ชันที่ใช้บ่อยๆในการค่าประมาณ Big-O

2.1.10 การทดสอบประสิทธิภาพ (Performance Test) [10]

การทดสอบประสิทธิภาพ (Performance testing) เป็นการทดสอบอย่างหนึ่งใน non-functional testing คือเป็นการทดสอบประสิทธิภาพของซอฟต์แวร์ที่ถูกพัฒนาขึ้นมา เช่น ทดสอบว่า

ระบบที่ถูกพัฒนาขึ้นมานั้นสามารถรองรับการทำงานหนักได้ดีมากน้อยเท่าใด เมื่อมีผู้ใช้งานจำนวนมาก ซอฟต์แวร์นั้นมีการตอบสนองเป็นอย่างไร

การทดสอบประสิทธิภาพสามารถแบ่งออกได้หลายอย่างตาม scenario

- Load testing คือการทดสอบซอฟต์แวร์หรือระบบว่า ระบบจะมีความเร็วมากน้อยแค่ไหน ภายใต้สภาวะและขนาดของภาระที่คาดว่าจะเกิดขึ้นจริง เช่น หากมีผู้ใช้งานเข้ามาใช้ระบบพร้อมกัน 100 คน ระบบจะตอบสนองเร็วหรือช้าแค่ไหน
- Stress testing คือ การทดสอบระบบที่นอกเหนือจากการทำงานปกติ เพื่อทดสอบความเสถียรในความพร้อมและการจัดการข้อผิดพลาด เมื่อระบบมีการทำงานหนัก
- Spike testing คือ การทดสอบระบบเมื่อมีการเพิ่มจำนวนผู้ใช้งานอย่างรวดเร็ว
- Soak testing หรือ Endurance testing คือ การทดสอบระบบว่า ระบบยังสามารถทำงานได้ดีหรือไม่ เมื่อมีการใช้ทำงานในเวลานาน throughput and/or response times ยังดีเหมือนกับตอนเริ่มต้นหรือไม่
- Capacity testing คือ การทดสอบเพื่อกำหนดหาว่า จะมีผู้ใช้กี่คนที่ระบบสามารถรองรับได้ โดยที่ระบบสามารถยังทำงานได้
- Recovery testing คือ การทดสอบระบบว่า ระบบสามารถฟื้นตัวจากการล่มได้เร็วหรือดีแค่ไหน
- Smoke testing คือ การเริ่มต้นทดสอบระบบในการทดสอบประสิทธิภาพ เพื่อดูว่า การระบบสามารถทำงานได้ปกติในสภาวะปกติ
- Volume testing คือ การทดสอบระบบโดยการใช้จำนวนข้อมูล เพื่อแสดงให้เห็นว่า จำนวนข้อมูลเท่าไหร่ที่ระบบไม่สามารถรองรับได้
- Network Sensitivity testing คือ การทดสอบขีดจำกัดของ WAN และ การทำงานของ network สามารถที่จะคาดการณ์ผลกระทบในส่วนของ WAN และ การสื่อสารบน bandwidth
- Scalability testing คือ การทดสอบเพื่อวัดความสามารถในการประยุกต์ใช้เมื่อนำไปใช้กับระบบที่ใหญ่ขึ้น หรือ ระบบอื่นๆที่จะทำไปใช้

2.2 เครื่องมือ

2.2.1 Eclipse

Eclipse [13] คือ โปรแกรมที่ใช้สำหรับพัฒนาภาษา Java ซึ่งโปรแกรม Eclipse เป็นโปรแกรมหนึ่งที่ใช้ในการพัฒนา Application Server ได้อย่างมีประสิทธิภาพ และเนื่องจาก Eclipse เป็นซอฟต์แวร์ OpenSource ที่พัฒนาขึ้นเพื่อใช้โดยนักพัฒนาเอง ทำให้ความก้าวหน้าในการพัฒนาของ Eclipse เป็นไปอย่างต่อเนื่องและรวดเร็ว

Eclipse มีองค์ประกอบหลักที่เรียกว่า Eclipse Platform ซึ่งให้บริการพื้นฐานหลักสำหรับรวบรวมเครื่องมือต่างๆจากภายนอกให้สามารถเข้ามาทำงานร่วมกันในสภาพแวดล้อมเดียวกัน และมีองค์ประกอบที่เรียกว่า Plug-in Development Environment (PDE) ซึ่งใช้ในการเพิ่มความสามารถในการพัฒนาซอฟต์แวร์มากขึ้น เครื่องมือภายนอกจะถูกพัฒนาในรูปแบบที่เรียกว่า Eclipse plug-ins ดังนั้นหากต้องการให้ Eclipse ทำงานใดเพิ่มเติม ก็เพียงแต่พัฒนา plugin สำหรับงานนั้นขึ้นมา และนำ Plug-in นั้นมาติดตั้งเพิ่มเติมให้กับ Eclipse ที่มีอยู่เท่านั้น Eclipse Plug-in ที่มีมาพร้อมกับ Eclipse เมื่อเรา download มาครั้งแรกก็คือองค์ประกอบที่เรียกว่า Java Development Toolkit (JDT) ซึ่งเป็นเครื่องมือในการเขียนและ Debug โปรแกรมภาษา Java

ข้อดีของโปรแกรม Eclipse คือ ติดตั้งง่าย สามารถใช้ได้กับ J2SDK ได้ทุกเวอร์ชั่น รองรับภาษาต่างประเทศอีกหลายภาษา มี plugin ที่ใช้เสริมประสิทธิภาพของโปรแกรม สามารถทำงานได้กับไฟล์หลายชนิด เช่น HTML, Java, C, JSP, EJB, XML และ GIF และที่สำคัญเป็นฟรีแวร์ (ให้ใช้งานได้ 90 วัน ถ้าจะใช้งานเต็มประสิทธิภาพต้องเสียค่าใช้จ่ายภายหลัง) ใช้งานได้กับระบบปฏิบัติการ Windows, Linux และ Mac OS

2.2.2 EGit

EGit [14] เป็นโปรแกรมที่ให้บริการ Git version control บน Eclipse Git คือ Version Control ตัวหนึ่ง ซึ่งเป็นระบบที่มีหน้าที่ในการจัดเก็บการเปลี่ยนแปลงของไฟล์ในโปรเจ็ค มีการ backup code สามารถที่จะเรียกดูหรือย้อนกลับไปดูเวอร์ชั่นต่างๆ ของโปรเจ็คได้ หรือแม้แต่ดูว่าไฟล์นั้นๆ ใครเป็นคนเพิ่มหรือแก้ไข หรือว่าจะดูว่าไฟล์นั้นๆ ถูกเขียนโดยใครบ้างก็สามารถทำได้ ฉะนั้น Version Control จึงเหมาะอย่างยิ่งสำหรับนักพัฒนาไม่ว่าจะเป็นคนเดียว โดยเฉพาะอย่างยิ่งจะมีประสิทธิภาพมากหากเป็นการพัฒนาเป็นทีม

บทที่ 3
วิธีการดำเนินงาน

ในการปฏิบัติงานสหกิจครั้งนี้ ได้รับมอบหมายให้ทำการพัฒนา การตัดคำภาษาไทยแบบคีย์เวิร์ดบนระบบการค้นหาข้อมูล (Thai word segment keyword on search engine) โดยใช้โครงสร้างข้อมูลแบบทรัย (Trie Structure) มาเป็นโครงสร้างข้อมูลหลักที่ใช้ในการสืบค้นคำ และทำการทดสอบเพื่อวัดประสิทธิภาพการทำงานของแต่ละโครงสร้าง โดยมีขั้นตอนการดำเนินงาน ดังนี้

3.1 ปัญหาที่เกิดขึ้น

คำในภาษาไทยมักนิยมเขียนเป็นประโยคหรือบทความยาวๆ โดยไม่มีการเว้นช่องว่างระหว่างคำ ซึ่งแตกต่างจากภาษาอังกฤษที่มีช่องว่างระหว่างคำอย่างชัดเจน จึงทำให้การตัดคำในภาษาไทยค่อนข้างมีความซับซ้อนและเกิดปัญหาไม่เจอผลลัพธ์ของคำทุกคำที่มีความหมายในประโยค รวมไปถึงกรณีที่พบคำสะกดผิดปรากฏในประโยคเหล่านั้น ผลลัพธ์การตัดคำจึงไม่แสดงออกมา

โปรแกรมการตัดคำภาษาไทย LexToPlus เป็นโปรแกรมตัดคำภาษาไทยแบบอิงพจนานุกรม (Dictionary based) โดยใช้เทคนิควิธีการตัดคำแบบยาวที่สุด (Longest matching) [15] สำหรับการตัดคำ การตัดคำที่เหมาะสมกับงานที่นำไปใช้หรือผลลัพธ์การตัดคำที่รองรับการทำ Normalize แล้วจะแสดงผลลัพธ์ออกมาให้เพียงคำตอบเดียว ซึ่งคล้ายกับการตัดคำทั่วไปที่ยังไม่สามารถรองรับการตัดคำสำหรับการทำดัชนีแบบอินเวิร์ทได้ เนื่องจากการทำดัชนีแบบอินเวิร์ทต้องการคำทุกคำที่เป็นไปได้ในประโยค เพื่อผลลัพธ์ที่มีประสิทธิภาพสำหรับการค้นหา

3.2 แนวทางการแก้ไขปัญหา

จากปัญหาที่เกิดขึ้นตามที่กล่าวมาข้างต้นนั้น เพื่อการแก้ไขปัญหาที่ถูกต้องตามวัตถุประสงค์ของโครงงานและจุดมุ่งหมายในการนำไปใช้งาน จึงมีแนวทางในการแก้ปัญหา ดังนี้

3.2.1 ศึกษาทฤษฎีและโครงสร้างข้อมูลที่เกี่ยวข้อง ได้แก่

- โครงสร้างข้อมูลแบบทรัย (Trie Structure)
- ความสัมพันธ์เวียนเกิด (Recurrence Relation)
- การค้นตามแนวกว้าง (Breadth first search)
- การค้นหาตามแนวลึก (Depth first search)
- คิว (Queue)
- การขยายและการจำกัดเขต (Branch and bound)
- การวิเคราะห์อัลกอริทึม (Algorithm Analysis)

- การแปลงโครงสร้างข้อมูลเชิงวัตถุให้เป็นสายอักขระ (Serialization)
- การทดสอบประสิทธิภาพ (Performance Test)

ซึ่งรวมไปถึงเทคนิควิธีการตัดคำแบบยาวที่สุดและวิธีการย้อนรอยกลับ (Backtracking) [15] โดยทฤษฎีและโครงสร้างข้อมูลที่ได้ทำการศึกษาทั้งหมดนี้ ได้ถูกนำไปใช้ในการดำเนินงาน เพื่อให้โครงการสามารถดำเนินงานตามขอบเขตที่กำหนดไว้

3.2.2 ความสัมพันธ์ของปัญหา

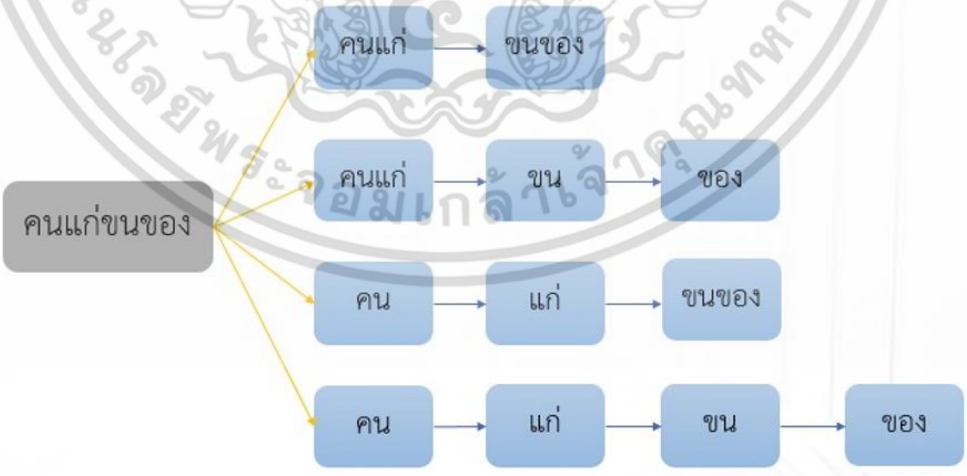
จากปัญหาที่กล่าวมาข้างต้น ทำให้ผู้พัฒนาคิดความสัมพันธ์ของปัญหาที่เกิดขึ้น โดยแบ่งรูปแบบการตัดคำภาษาไทยแบบพจนานุกรมออกเป็น 2 รูปแบบ คือ 1. แบบ Safe segmentation และ 2. แบบ Unsafe segmentation

1. Safe segmentation

เป็นวิธีการตัดคำที่ให้ผลลัพธ์เป็นทุกคำของการตัดคำ ซึ่งมีการทำงานแบบย้อนรอยและการตัดคำโดยให้ผลลัพธ์ของทุกความเป็นไปได้ในแต่ละประโยคหรือบทความนั้น แต่เมื่อพบคำที่สะกดผิด การตัดคำรูปแบบนี้จะไม่สามารถแสดงผลลัพธ์ใดๆ การตัดคำแบบ Safe segmentation มีคุณสมบัติ ดังนี้

- ประโยคหรือบทความสามารถแทนด้วยคำภาษาไทยได้ทั้งหมด
- สามารถแสดงทุกรูปแบบของการจัดเรียงที่เป็นไปได้

ตัวอย่าง ผลลัพธ์ของการตัดคำแบบ Safe segmentation แสดงดังรูปที่ 3.1



รูปที่ 3.1 ตัวอย่างการตัดคำแบบ Safe segmentation

ความสัมพันธ์เวียนเกิดการตัดคำแบบ Safe segmentation แสดงดังสมการ (Recurrence relation)

$$\text{Segment}_{\text{trie}}(\Sigma, \Omega) = \bigcup_{\rho \in Prefix_{trie,\Omega}} Segment_{trie}\left(\Sigma_{Append(\rho)}, \Omega_{Substring(\rho)}\right)$$

ให้เงื่อนไขเริ่มต้น (Initial condition) = $\text{Segment}_{\text{trie}}(\Sigma, \emptyset) = \boldsymbol{\Sigma}$ และคำอธิบายสัญลักษณ์เป็นดังที่แสดงในตารางที่ 3.1

ตารางที่ 3.1 สัญลักษณ์สมการ

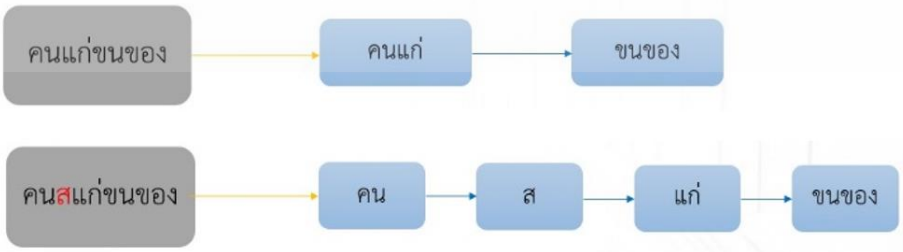
สัญลักษณ์	คำอธิบาย
Σ	ลำดับของคำ
Ω	ประโยคหรือบทความที่รับมา
prefix	คำขึ้นต้นของประโยคทั้งหมดในบทความนั้นๆ
$\emptyset$	เซตว่าง

2. Unsafe segmentation

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

- ประโยคหรือบทความไม่สามารถแทนด้วยคำไทยได้ทั้งหมด
- ข้ามอักขระทีละอักษรเมื่อพบตัวอักขระที่สะกดผิด และแสดงผลลัพธ์คำที่ยาวที่สุดของแต่ละคำในประโยคเพียงรูปแบบเดียว

ตัวอย่าง ผลลัพธ์ของการตัดคำแบบ Unsafe segmentation แสดงดังรูปที่ 3.2



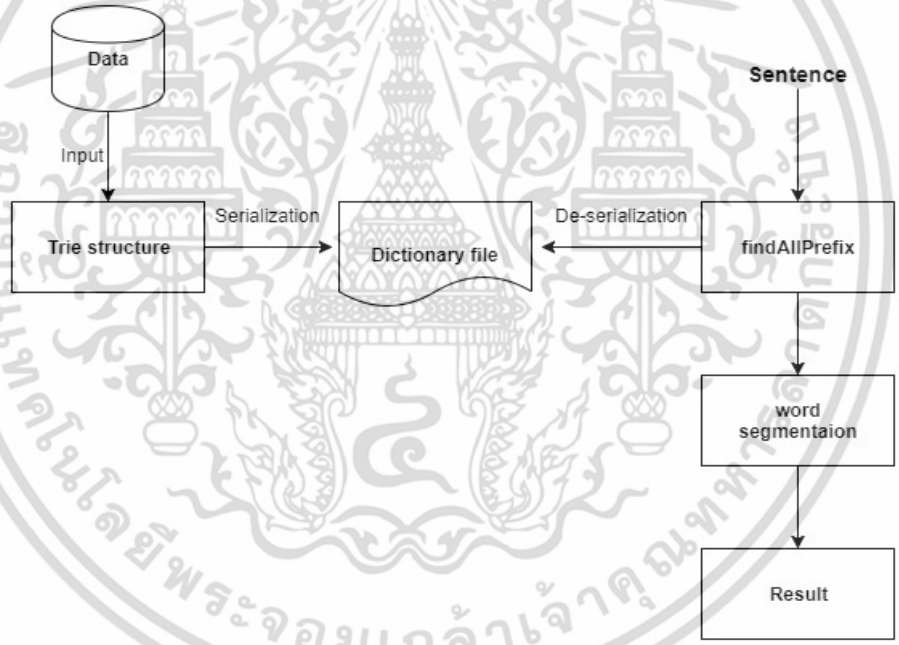
รูปที่ 3.2 ตัวอย่างการตัดคำแบบ Unsafe segmentation

3.2.3 พัฒนาฟังก์ชันที่ตอบสนองความต้องการ

ในการตัดคำภาษาไทยผู้พัฒนานำโครงสร้างข้อมูลแบบทรัย (Trie Structure) เป็นโครงสร้างข้อมูลหลักในการสืบค้นคำ เนื่องจากโครงสร้างข้อมูลแบบทรัยขาดฟังก์ชันที่ใช้สำหรับค้นหาคำขึ้นต้นทั้งหมดของแต่ละคำในประโยค เพื่อให้ตรงตามความต้องการในการนำไปใช้งานในโปรแกรมจึงได้คิดค้นฟังก์ชันในการค้นหาคำขึ้นต้นของประโยค เพื่อตอบสนองความต้องการโดยเรียกฟังก์ชันนี้ว่า findAllPrefix มาใช้ในโปรแกรมการตัดคำภาษาไทย ได้อย่างมีประสิทธิภาพ

3.3 ขั้นตอนการดำเนินงานในการแก้ไขปัญหา

ขั้นตอนการดำเนินงานสำหรับการตัดคำภาษาไทยแบบคีย์เวิร์ดบนระบบการค้นหาข้อมูล แสดงแผนภาพการทำงานดังรูปที่ 3.3

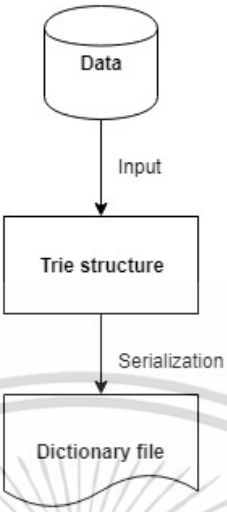


รูปที่ 3.3 แผนภาพการทำงานการตัดคำภาษาไทยแบบคีย์เวิร์ดบนระบบการค้นหาข้อมูล

ขั้นตอนการดำเนินงานได้แบ่งการทำงานออกเป็น 2 ส่วนหลัก คือ

1. **ขั้นตอนการสร้างพจนานุกรม**

สำหรับขั้นตอนนี้นำคลังคำที่ทำการเตรียมไว้ เพิ่มรายการคำลงไปในโครงสร้างทรัยแล้วทำการแปลงโครงสร้างทรัยที่บรรจุคลังคำให้เป็นสายอักขระ เป็นขั้นตอนการเตรียมพจนานุกรมสำหรับการตัดคำภาษาไทย ซึ่งขั้นตอนนี้ประกอบด้วยหัวข้อต่อไปนี้และแสดงแผนภาพการทำงานดังรูปที่ 3.4



รูปที่ 3.4 แผนภาพการสร้างพจนานุกรมสำหรับการตัดคำภาษาไทย

3.3.1 คลังคำ

คลังคำที่นำมาใช้เป็นพจนานุกรมการตัดคำภาษาไทยแบบคีย์เวิร์ดบนระบบการค้นหาข้อมูลเป็นคลังคำของบริษัท ทิงค์เน็ต จำกัด ที่ได้ทำการรวบรวมรายการคำ เพื่อนำคลังคำมาใช้ในการศึกษาและพัฒนาภายในองค์กร

3.3.2 วิเคราะห์หลักการทำงานของโครงสร้างข้อมูลแบบทรัย

ผู้พัฒนาได้ศึกษาค้นคว้าและวิเคราะห์โครงสร้างข้อมูลสำเร็จรูปแบบทรัย (Trie Structure) 4 โครงสร้าง โดยในที่นี้เรียกว่า TrieA, TrieB, TrieC, และ TrieD เพื่อไม่ให้เกิดความสับสนในการนำไปใช้งาน รวมทั้งนำฟังก์ชัน findAllPrefix เพิ่มลงในโครงสร้างข้อมูลแบบทรัย แล้วคัดเลือกโครงสร้างข้อมูลจากการทดสอบประสิทธิภาพในแง่เวลาของการทำงาน ซึ่งโครงสร้างข้อมูลแบบทรัยทั้ง 4 โครงสร้าง มีด้วยกัน 3 รูปแบบ แสดงรายละเอียด ดังนี้

<u>TrieA</u>

TrieA มีโครงสร้างการทำงานแบบลิ้งค์ลิสต์ (Linklist) ซึ่งมีวิธีการเก็บข้อมูลอย่างต่อเนื่องของอิลิเม้นต์ต่าง ๆ โดยมีพอยเตอร์เป็นตัวเชื่อมต่อแต่ละอิลิเม้นท์ เรียกว่าโหนด (Node) ซึ่งในแต่ละโหนดจะประกอบไปด้วย 2 ส่วน คือ Data จะเก็บข้อมูลของอิลิเม้นท์ อาจจะเป็นรายการเดียวหรือเป็นเรคคอร์ดก็ได้และส่วนที่สองคือ Link Field จะทำหน้าที่เก็บตำแหน่งของโหนดถัดไปในลิสต์ โหนดสุดท้ายจะเก็บค่า Null ซึ่งไม่ได้ชี้ไปยังตำแหน่งใด ๆ เป็นตัวบอกการสิ้นสุดของลิสต์

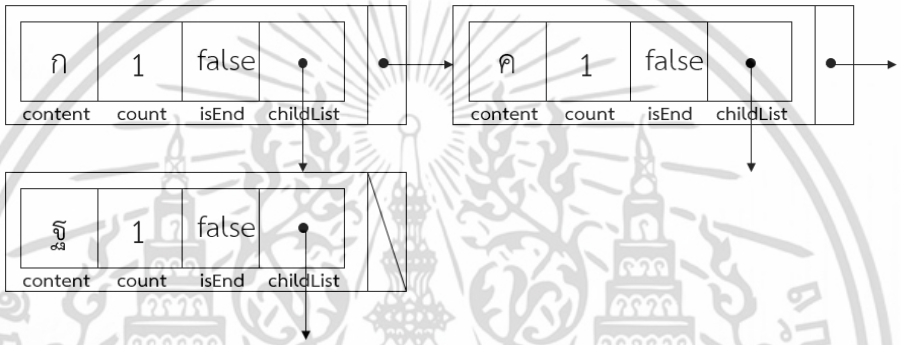
โดยส่วนของ Data จะประกอบไปด้วย

- content คือ ตัวอักษร
- count คือ จำนวนคำทั้งหมดที่ขึ้นต้นด้วย content นั้นๆ
- isEnd คือ การตรวจสอบว่า content เป็นตัวอักษรตัวสุดท้ายของคำหรือไม่
- childList คือ ลิงก์ที่เชื่อมโยงไปยังตัวอักษรตัวถัดไปของแต่ละคำ

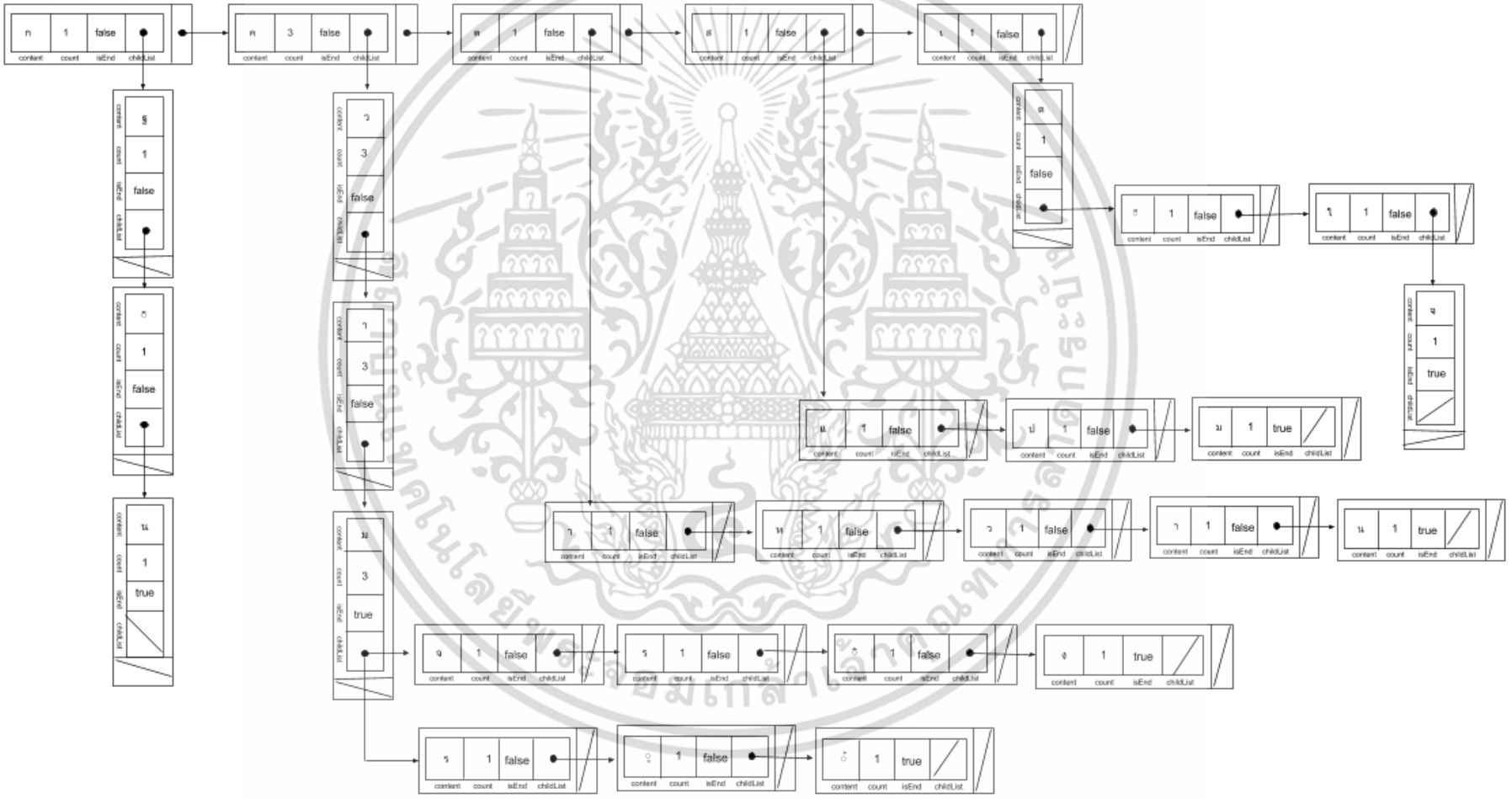
และ Link Field จะทำหน้าที่เก็บตำแหน่งตัวอักษรขึ้นต้นของคำใหม่

โครงสร้างการทำงานของ TrieA แสดงตัวอย่างได้ดังรูปที่ 3.5

รูปที่ 3.5 TrieNode ที่มีการทำงานแบบลิ้งค์ลิสต์



<u>ตัวอย่าง</u> รายการคำใน TrieA ได้แก่ กฐิน ความ ความจริง ความรู้ ตาหวาน สแปม เต็มใจ แสดงโครงสร้างการทำงานดังรูปที่ 3.6



รูปที่ 3.6 ตัวอย่างโครงสร้างการทำงานของ TrieA

TrieB

TrieB มีโครงสร้างการทำงานแบบแฮชแม็พ (HashMap) ซึ่งมีรูปแบบการจัดเก็บค่าตัวแปรที่อยู่ในรูปแบบของ Key และ Value โดยแฮชแม็พสามารถเพิ่มสมาชิกได้ไม่จำกัดจำนวนและไม่ต้องทราบถึงจำนวนขนาดหรือ Size สมาชิกล่วงหน้า กล่าวคือ สามารถเพิ่มได้เรื่อย ๆ จนเพียงพอแต่ความต้องการใช้งาน แต่การจัดเก็บจนมาเกินไปก็จะมีผลต่อหน่วยความจำ (memory) และความเร็วในการทำงานเช่นเดียวกัน เพราะฉะนั้นการใช้ตัวแปรเหล่านี้จะต้องใช้ให้เหมาะสม ถ้าต้องการจัดเก็บข้อมูลประเภท Object , Byte หรือ Bitmap จำเป็นต้องใช้หน่วยความจำจำนวนมากในการจัดเก็บ แต่ถ้าเป็นข้อมูลประเภท String หรือ Int (Number) จะสามารถเก็บได้จำนวนมาก คุณสมบัติอีกประการของแฮชแม็พคือ ยอมให้มีค่าว่าง (null) เป็นทั้ง Key และ Value และแฮชแม็พเป็นประเภท unsynchronized ดังนั้น TrieNode จะประกอบไปด้วย

character คือ ตัวอักษร
children คือ แฮชแม็พที่จัดเก็บตัวอักษรตัวถัดไปของแต่ละคำ
isWord คือ การตรวจสอบว่า character เป็นตัวอักษรตัวสุดท้ายของแต่ละคำหรือไม่

ซึ่ง children เป็นแฮชแม็พที่มีลิงก์เชื่อมโยงตัวอักษรของคำ ประกอบด้วย

- Key
  - Value คือ ตัวอักษร
- Value
  - character คือ ตัวอักษร
  - children คือ แฮชแม็พที่จัดเก็บตัวอักษรตัวถัดไปของแต่ละคำ
  - isWord คือ การตรวจสอบว่า character เป็นตัวอักษรตัวสุดท้ายของแต่ละคำหรือไม่

TrieC

TrieC มีโครงสร้างการทำงานแบบแฮชแม็พ (HashMap) เช่นเดียวกับ TrieB แต่มีโครงสร้างการพัฒนาและฟังก์ชันการทำงานที่แตกต่างกัน จึงทำให้การวัดประสิทธิภาพในแง่ของเวลาการทำงานมีความแตกต่างกันโดยสิ้นเชิง ดังนั้น TrieNode จะประกอบไปด้วย

bIsEnd คือ การตรวจสอบว่า value เป็น character ตัวสุดท้ายของคำหรือไม่
children คือ แฮชแม็พที่จัดเก็บตัวอักษรตัวถัดไปของแต่ละคำ
value คือ ตัวอักษร

ซึ่ง children เป็นแฮชแม็พเช่นเดียวกับ TrieB มีลิงก์เชื่อมโยงตัวอักษรของคำ ประกอบด้วย

- Key

  Value คือ ตัวอักษร

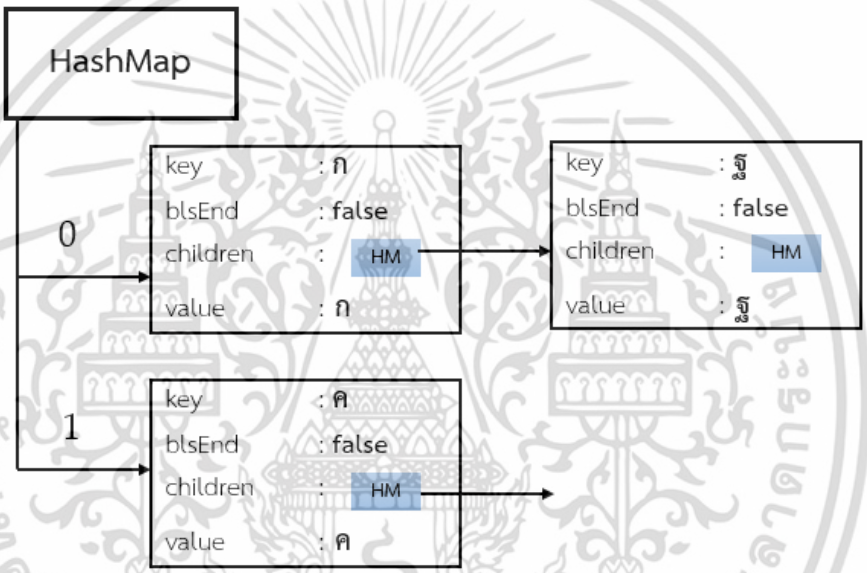
- Value

  blsEnd คือ การตรวจสอบว่า value เป็น character ตัวสุดท้ายของคำหรือไม่

  children คือ แฮชแม็พที่จัดเก็บตัวอักษรตัวถัดไปของแต่ละคำ

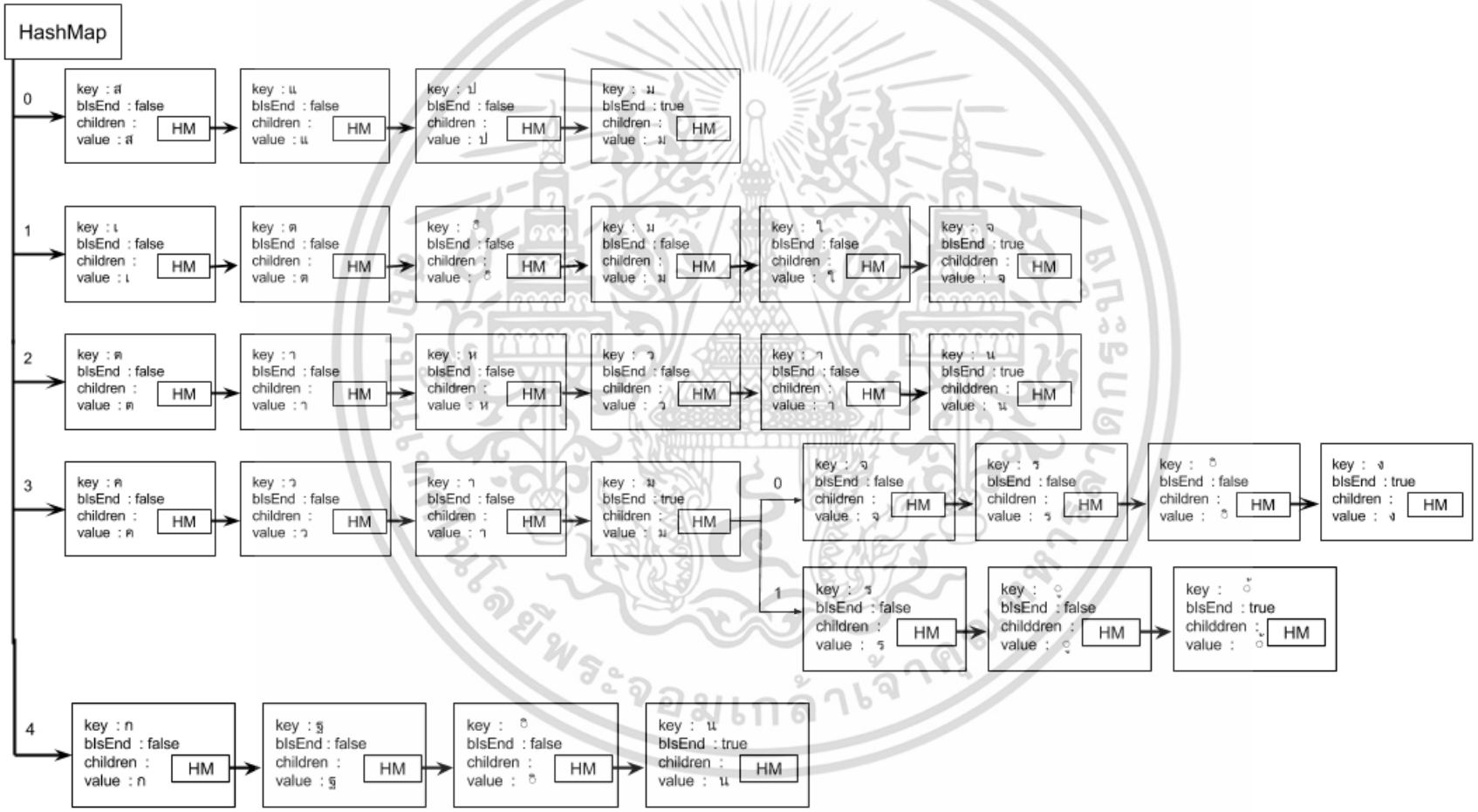
  value คือ ตัวอักษร

โครงสร้างการทำงานของ TrieNode แสดงตัวอย่างได้ดังรูปที่ 3.7



รูปที่ 3.7 TrieNode ที่มีการทำงานแบบแฮชแม็พ

<u>ตัวอย่าง</u> รายการคำใน TrieB และ TrieC ได้แก่ กฐิน ความ ความจริง ความรู้ ตาหวาน สแปม เต็มใจ แสดงโครงสร้างการทำงานดังรูปที่ 3.8



รูปที่ 3.8 ตัวอย่างโครงสร้างการทำงานของ TrieB และ TrieC

TrieD

TrieD มีโครงสร้างการทำงานแบบ Node tree จะประกอบด้วยโหนด (node) ซึ่งเป็นส่วนที่เก็บข้อมูล ใน Node tree จะประกอบไปด้วยโหนดราก (root node) เพียงหนึ่งโหนด แล้วรูทโหนดสามารถแตกโหนดออกเป็นโหนดย่อยๆ ได้อีกหลายโหนดเรียกว่าโหนดลูก (Child node) เมื่อมีโหนดลูกแล้วโหนดลูกก็ยังสามารถแสดงเป็นโหนดพ่อแม่ (Parent Node) โดยการแตกโหนดออกเป็นโหนดย่อยๆได้อีก ซึ่งโหนดรากจะเป็นตัวอักษรตัวแรกของคำแรกที่ทำการเพิ่มข้อมูลลงในโครงสร้าง

ดังนั้น TrieNode จะประกอบไปด้วย

character คือ ตัวอักษร

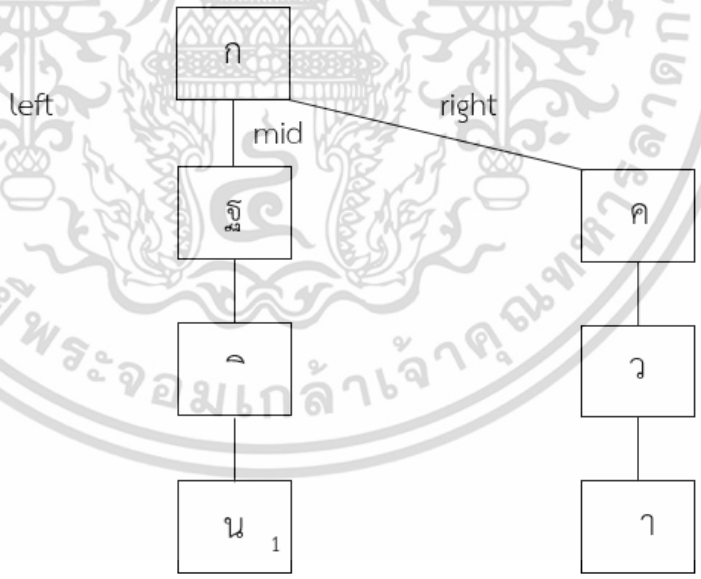
left คือ ท่องโหนดไปทางซ้าย

mid คือ ท่องโหนดไปตรงข้าม

right คือ ท่องโหนดไปทางขวา

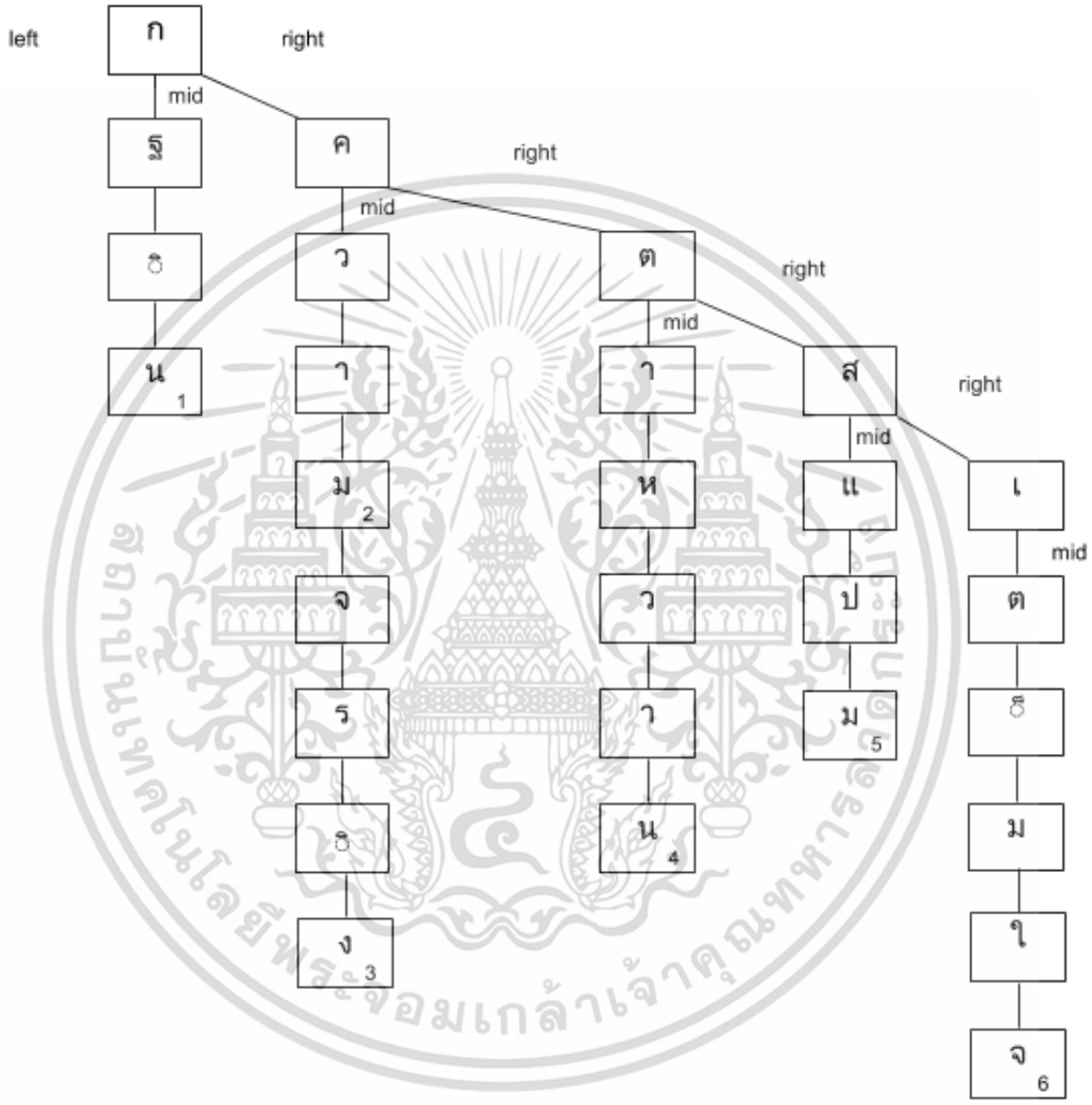
value คือ ค่าที่เกี่ยวข้องกับคำ เพื่อบ่งบอกว่าคำๆนี้ สิ้นสุดที่ตัวอักษรใด โดยตรวจสอบจากค่า value หากยังไม่ใช่ตัวอักษรตัวสุดท้ายในแต่ละคำนั้น ค่า value จะมีค่าว่าง(null)

โครงสร้างการทำงานของ TrieNode แสดงตัวอย่างได้ดังรูปที่ 3.9



รูปที่ 3.9 TrieNode ที่มีการทำงานแบบ Node tree

<u>ตัวอย่าง</u> รายการคำใน TrieD ได้แก่ กฐิน ความ ความจริง ความรู้ ตาหวาน สแปม เต็มใจ แสดงโครงสร้างการทำงานดังรูปที่ 3.10



รูปที่ 3.10 ตัวอย่างโครงสร้างการทำงานของ TrieD

3.3.3 การแปลงโครงสร้างข้อมูลเชิงวัตถุให้อยู่ในรูปสายอักขระ

การแปลงโครงสร้างข้อมูลแบบทรัยที่มีคลังคำอยู่ภายในโครงสร้างทรัยให้อยู่ในรูปของสายอักขระ เพื่อให้สามารถเก็บวัตถุในรูปของไฟล์ได้ และเพื่อความสะดวกของการเตรียมโครงสร้างข้อมูลแบบทรัยที่มีคลังคำอยู่ภายใน นำไปใช้สำหรับการตัดคำภาษาไทย

ผู้พัฒนาทำการแปลงโครงสร้างข้อมูลแบบทรัยที่มีคลังคำอยู่ภายในเก็บไว้ เมื่อมีการตัดคำภาษาไทยจะทำการเรียกสายอักขระให้กลับเป็นโครงสร้างทรัย (De-serialization) ดังเดิม ขั้นตอนนี้ทำให้ไม่ต้องเพิ่มคลังคำเข้าไปในโครงสร้างข้อมูลแบบทรัยทุกครั้งที่มีการตัดคำภาษาไทย เพียงเรียกใช้งานสายอักขระให้กลับเป็นโครงสร้างทรัยเมื่อต้องการตัดคำภาษาไทย ซึ่งการแปลงโครงสร้างข้อมูลเชิงวัตถุให้อยู่ในรูปสายอักขระจะทำการเลือก java serializable library ที่มีความเร็วที่สุดเมื่อทำงานร่วมกับโครงสร้างข้อมูลแบบทรัย จากการทดสอบวัดประสิทธิภาพแง่เวลาของแต่ละไลบรารี่ที่มีการทดสอบไว้แล้ว จึงได้ java serializable library มาทั้งหมด 8 ไลบรารี่ที่นำมาใช้งานร่วมกับโครงสร้างข้อมูลแบบทรัย

Java serializable library แบ่งออกเป็น 2 ประเภท

1. Automatic Serialization with Trie structure คือ การแปลงโครงสร้างข้อมูลเชิงวัตถุให้อยู่ในรูปสายอักขระ ที่สามารถเรียกใช้ฟังก์ชันการ serialization และ de-serialization โดยไม่ต้องทำการ implement serialization ของแต่ละไลบรารี่ลงโครงสร้างข้อมูลแบบทรัย มีทั้งหมด 5 ไลบรารี่ ได้แก่

1.1. GSON คือ ไลบรารี่ที่ทำการแปลงข้อมูล Java objects เป็น JSON และทำการแปลง JSON เป็น Java objects การแปลงโครงสร้างข้อมูลเชิงวัตถุให้อยู่ในรูปสายอักขระของ GSON แสดงดังรูปที่ 3.11

```
public class GSONSerialization<T> {
        private static final ConfigPath path = new ConfigPath(); //path for read and write file.
        private Gson gson = new Gson();
        //Serialization
        public void convertFromObj(T t, String filename) throws IOException {
                try (FileWriter writer = new FileWriter(path.path() + filename)) {
                        gson.toJson(t, writer);
                } catch (IOException e) {
                        throw new IOException();
                }
        }
        //De-serialization
        public T convertToObj(T t, String filename) {
                T trie = null;
                try (Reader reader = new FileReader(path.path() + filename)) {
                        trie = (T) gson.fromJson(reader, t.getClass());
                } catch (IOException e) {
                        e.printStackTrace ();
                }
                return trie;
        }
}
```

รูปที่ 3.11 การแปลงโครงสร้างข้อมูลเชิงวัตถุให้อยู่ในรูปสายอักขระของ GSON

1.2. Protobuf

- พัฒนาและใช้งานโดย Google
- รองรับภาษา Java, C++ และ Python
- มีความยืดหยุ่นและมีประสิทธิภาพ

การแปลงโครงสร้างข้อมูลเชิงวัตถุให้อยู่ในรูปสายอักขระของ Protobuf แสดงดังรูปที่ 3.12

```
public class ProtobufSerialization<T> {
        private static final ConfigPath path = new ConfigPath();
        static final int bufferSize = 2048;
        static final ThreadLocal<LinkedBuffer> localBuffer = new ThreadLocal<LinkedBuffer>()  {
                public LinkedBuffer initialValue() { return LinkedBuffer.allocate(bufferSize); }
        };
        private static LinkedBuffer getApplicationBuffer() {
                return localBuffer.get();
        }
        //Serialization
        public void convertFromObj(T t, String filename) throws FileNotFoundException,
           IOException {
                Schema<T> schema = (Schema<T>) RuntimeSchema.getSchema(t.getClass());
                OutputStream out = new FileOutputStream(path.path() + filename);
                convertFromObj (t, schema, buffer, out);
        }
        private void convertFromObj(T t, Schema<T> schema,  LinkedBuffer buffer,
          OutputStream out) throws FileNotFoundException, IOException {
                 try {   ProtobufIOUtil.writeTo(out, t, schema, buffer);
                } finally { buffer.clear(); }
        }
        //De-serialization
        public T convertToObj(T t, String filename) throws FileNotFoundException,
           ClassNotFoundException, IOException {
                Schema<T> schema = (Schema<T>) RuntimeSchema.getSchema(t.getClass());
                InputStream in = new FileInputStream (path.path() + filename);
                return convertToObj(schema, buffer, in);
        }
        private T convertToObj(Schema<T> schema, LinkedBuffer buffer, InputStream in) throws
          FileNotFoundException, IOException, ClassNotFoundException {
                T trie = schema.newMessage();
                ProtobufIOUtil.mergeFrom(in, trie, schema, buffer);
                 return trie;
        }
}
```

รูปที่ 3.12 การแปลงโครงสร้างข้อมูลเชิงวัตถุให้อยู่ในรูปสายอักขระของ Protobuf

1.3. Protostuff เป็นไลบรารี่ที่มีการรองรับการทำงานแบบ forward-backward การแปลงโครงสร้างข้อมูลเชิงวัตถุให้อยู่ในรูปสายอักขระของ Protostuff แสดงดังรูปที่ 3.13

```
public class ProtostuffSerialization<T> {
        private static final ConfigPath path = new ConfigPath();
        static final int bufferSize = 2048;
        static final ThreadLocal<LinkedBuffer> localBuffer = new ThreadLocal<LinkedBuffer>() {
                public LinkedBuffer initialValue() { return LinkedBuffer.allocate(bufferSize); }
        };
        private static LinkedBuffer getApplicationBuffer() {
                return localBuffer.get();
        }
        public void convertFromObj(T t, String filename) throws FileNotFoundException,
         IOException {
                Schema<T> schema = (Schema<T>) RuntimeSchema.getSchema(t.getClass());
                OutputStream out = new FileOutputStream(path.path() + filename);
                convertFromObj(t, schema, buffer, out);
        }
        private void convertFromObj(T t, Schema<T> schema, LinkedBuffer buffer,
         OutputStream out) throws FileNotFoundException, IOException {
                try { ProtostuffIOUtil.writeTo(out, t, schema, buffer);
                } finally { buffer.clear(); }
        }
        public T convertToObj(T t, String filename) throws FileNotFoundException,
         ClassNotFoundException, IOException {
                Schema<T> schema = (Schema<T>) RuntimeSchema.getSchema(t.getClass());
                InputStream in = new FileInputStream(path.path() + filename);
                return convertToObj(schema, buffer, in);
        }
        private T convertToObj(Schema<T> schema, LinkedBuffer buffer, InputStream in) throws
         FileNotFoundException, IOException, ClassNotFoundException {
                T trie = schema.newMessage();
                ProtostuffIOUtil.mergeFrom(in, trie, schema, buffer);
                return trie;
        }
}
```

รูปที่ 3.13 การแปลงโครงสร้างข้อมูลเชิงวัตถุให้อยู่ในรูปสายอักขระของ Protostuff

1.4. Protostuff-json คือ ไลบรารี่ของ protostuff ที่ทำการแปลงข้อมูล Java objects เป็น JSON และทำการแปลง JSON เป็น Java objects การแปลงโครงสร้างข้อมูลเชิงวัตถุให้อยู่ในรูปสายอักขระของ Protostuff-json แสดงดังรูปที่ 3.14

```
public class JsonSerialization<T> {
        private static final ConfigPath path = new ConfigPath();
        final int bufferSize = 2048;
        final ThreadLocal<LinkedBuffer> localBuffer = new ThreadLocal<LinkedBuffer>() {
                public LinkedBuffer initialValue() {return LinkedBuffer.allocate(bufferSize);
            }
        };
        private LinkedBuffer getApplicationBuffer() { return localBuffer.get(); }
        public void convertFromObj(T t, String filename) throws FileNotFoundException,
          IOException {
                Schema<T> schema = (Schema<T>) RuntimeSchema.getSchema(t.getClass());
                OutputStream out = new FileOutputStream(path.path() + filename);
                convertFromObj(t, schema, buffer, out);
        }
        private void convertFromObj(T t, Schema<T> schema, LinkedBuffer buffer,
         OutputStream out) throws FileNotFoundException, IOException {
                boolean numeric = false;
                JsonIOUtil.writeTo(out, t, schema, numeric, buffer);
        }
        public T convertToObj(T t, String filename) throws FileNotFoundException,
          ClassNotFoundException, IOException {
                Schema<T> schema = (Schema<T>) RuntimeSchema.getSchema(t.getClass());
                InputStream in = new FileInputStream(path.path() + filename);
                return convertToObj(schema, buffer, in);
        }
        private T convertToObj(Schema<T> schema, LinkedBuffer buffer, InputStream in) throws
          FileNotFoundException, IOException, ClassNotFoundException {
                boolean numeric = false;
                T trie = schema.newMessage();
                JsonIOUtil.mergeFrom(in, trie, schema, numeric, buffer);
                return trie;
        }
}
```

รูปที่ 3.14 การแปลงโครงสร้างข้อมูลเชิงวัตถุให้อยู่ในรูปสายอักขระของ Protostuff-json

1.5. Protostuff-xml คือ ไลบรารี่ของ protostuff ที่ทำการแปลงข้อมูล Java objects เป็น xml และทำการแปลง xml เป็น Java objects การแปลงโครงสร้างข้อมูลเชิงวัตถุให้อยู่ในรูปสายอักขระของ Protostuff-xml แสดงดังรูปที่ 3.15

```
public class XmlSerialization<T> {
        private static final ConfigPath path = new ConfigPath();
        static final int bufferSize = 2048;
        static final ThreadLocal<LinkedBuffer> localBuffer = new ThreadLocal<LinkedBuffer>() {
                public LinkedBuffer initialValue() { return LinkedBuffer.allocate(bufferSize); }
        };
        private static LinkedBuffer getApplicationBuffer() {
                return localBuffer.get();
        }
        public void convertFromObj(T t, String filename) throws FileNotFoundException,
          IOException {
                Schema<T> schema = (Schema<T>) RuntimeSchema.getSchema(t.getClass());
                OutputStream out = new FileOutputStream(path.path() + filename);
                convertFromObj(t, schema, buffer, out);
        }
        private void convertFromObj(T t, Schema<T> schema, LinkedBuffer buffer,
          OutputStream out) throws FileNotFoundException, IOException {
                XmlIOUtil.writeTo(out, t, schema);
        }
        public T convertToObj(T t, String filename) throws FileNotFoundException,
          ClassNotFoundException, IOException {
                Schema<T> schema = (Schema<T>) RuntimeSchema.getSchema(t.getClass());
                InputStream in = new FileInputStream(path.path() + filename);
                return convertToObj(schema, buffer, in);
        }
        private T convertToObj(Schema<T> schema, LinkedBuffer buffer, InputStream in) throws
          FileNotFoundException, IOException, ClassNotFoundException {
                T trie = schema.newMessage();
                XmlIOUtil.mergeFrom(in, trie, schema);
                return trie;
        }
}
```

รูปที่ 3.15 การแปลงโครงสร้างข้อมูลเชิงวัตถุให้อยู่ในรูปสายอักขระของ Protostuff-xml

2. Manual Serialization with Trie structure คือ การแปลงโครงสร้างข้อมูลเชิงวัตถุให้อยู่ในรูปสายอักขระ ที่สามารถเรียกใช้ฟังก์ชันการ serialization และ de-serialization โดยจะต้องทำการ implement serialization ของแต่ละไลบรารี่ลงโครงสร้างข้อมูลแบบทรัยก่อน มีทั้งหมด 3 ไลบรารี่ ได้แก่

2.1. FST จะเน้นเรื่องของ ความเร็ว ขนาดและความเข้ากันได้ของข้อมูล การแปลงโครงสร้างข้อมูลเชิงวัตถุให้อยู่ในรูปสายอักขระของ FST แสดงดังรูปที่ 3.16

```
public class FSTSerialization<T> {
        private static final ConfigPath path = new ConfigPath();
        //Serialization
        public void convertFromObj(String filename, T t) {
                try {
                        OutputStream stream = new FileOutputStream(path.path() + filename);
                        FSTObjectOutput out = new FSTObjectOutput(stream);
                        out.writeObject(t);
                        out.close();
                } catch (Exception e) { e.printStackTrace(); }
        }
        //De-serialization
        public T convertToObj(String filename) {
                T trie = null;
                try {
                        InputStream stream = new FileInputStream(path.path() + filename);
                        FSTObjectInput in = new FSTObjectInput(stream);
                        trie = (T)in.readObject();
                        in.close();
                } catch (Exception e) { e.printStackTrace(); }
                return trie;
        }
}
```

รูปที่ 3.16 การแปลงโครงสร้างข้อมูลเชิงวัตถุให้อยู่ในรูปสายอักขระของ FST

2.2. Kryo เป็นไลบรารี่ที่มีความเร็วและมีประสิทธิภาพในการทำ object graph serialization framework สำหรับจาวา การแปลงโครงสร้างข้อมูลเชิงวัตถุให้อยู่ในรูปสายอักขระจะทำการ implement KryoSerializable ในคลาสที่เราต้องการทำ serialization แสดงดังรูปที่ 3.17

```
public class KryoSerialization<T> {
        private static final ConfigPath path = new ConfigPath();
        private Kryo kryo = new Kryo();
        //Serialization
        public void convertFromObj(T t, String filename) {
                try (Output output = new Output(new FileOutputStream(path.path() + filename)))
                {
                        kryo.register(t.getClass());
                        kryo.writeObjectOrNull(output, t, t.getClass());
                        output.close();
                } catch (FileNotFoundException ex) {
                        Logger.getLogger(Kryo.class.getName()).log(Level.SEVERE, null, ex);
                }
        }
        //De-serialization
        public T convertToObj(T t, String filename) {
                T trie = null;
                try (Input input = new Input(new FileInputStream(path.path() + filename)))
                {
                        trie = (T)kryo.readObjectOrNull(input, t.getClass());
                        input.close();
                } catch (FileNotFoundException ex) {
                        Logger.getLogger(Kryo.class.getName()).log(Level.SEVERE, null, ex);
                }
                return trie;
        }
}
```

รูปที่ 3.17 การแปลงโครงสร้างข้อมูลเชิงวัตถุให้อยู่ในรูปสายอักขระของ Kryo

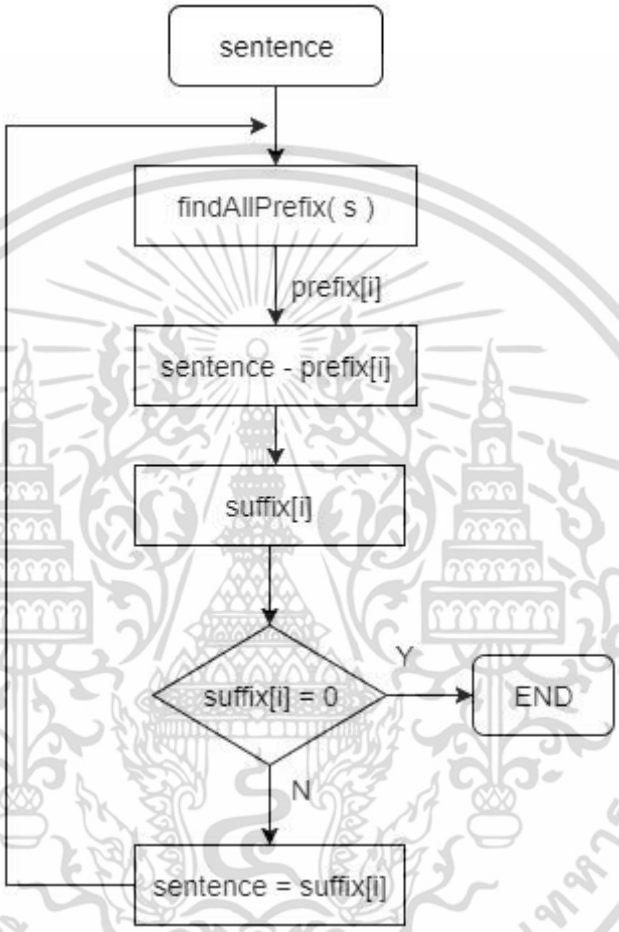
2.3 Java serialize คือ การแปลงโครงสร้างข้อมูลเชิงวัตถุให้อยู่ในรูปสายอักขระโดยการ implement java.io.Serializable ในคลาสที่เราต้องการทำ serialization แสดงดังรูปที่ 3.18

```
public class JavaSerialization<T> {
        //Serialization
        public void convertFromObj(T t, String filename) {
            try {
                ObjectOutputStream out = new ObjectOutputStream(new
                    FileOutputStream(path.path() + filename));
                        out.writeObject(t);
                        out.close();
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
        //De-serialization
        public T convertToObj(String filename) {
                T trie = null;
                try {
                        ObjectInputStream in = new ObjectInputStream(new
                            FileInputStream(path.path() + filename));
                        trie = (T)in.readObject();
                        in.close();
                } catch (Exception e) {
                        e.printStackTrace();
                }
                return trie;
        }
}
```

รูปที่ 3.18 การแปลงโครงสร้างข้อมูลเชิงวัตถุให้อยู่ในรูปสายอักขระของ Java serialize

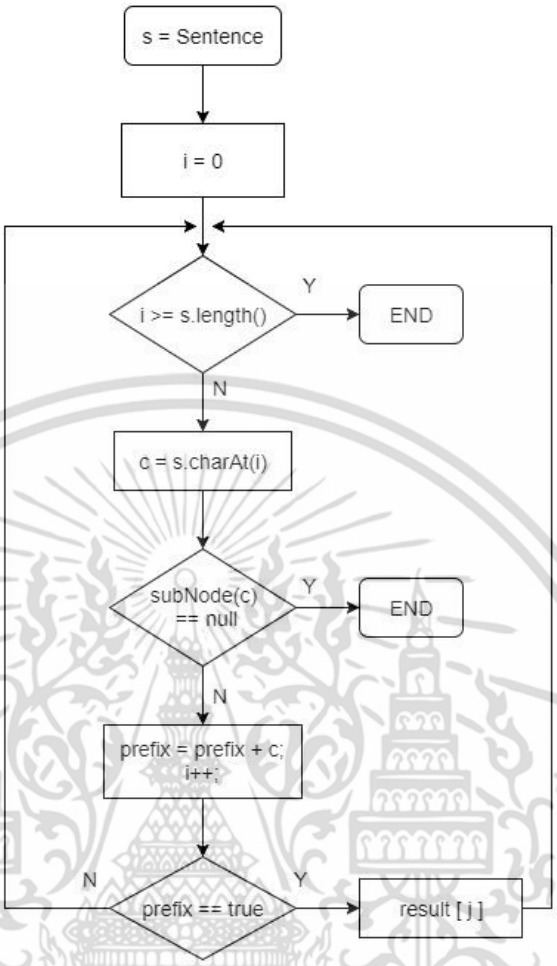
2. ขั้นตอนการตัดคำ

ขั้นตอนการตัดคำภาษาไทยจะทำการรับประโยคหรือบทความที่ต้องการตัดคำ เข้าสู่กระบวนการทำงานการตัดคำภาษาไทยแสดงแผนภาพการทำงานดังรูปที่ 3.19



รูปที่ 3.19 แผนภาพขั้นตอนการตัดคำภาษาไทย

ซึ่งภายในขั้นตอนการตัดคำภาษาไทยได้มีการพัฒนาฟังก์ชันค้นหาคำขึ้นต้นของแต่ละคำที่มีทั้งหมดในประโยคหรือบทความแสดงแผนภาพการทำงานดังรูปที่ 3.20 และการทำงานหลักสำหรับการตัดคำภาษาไทยประกอบด้วยหัวข้อต่อไปนี้

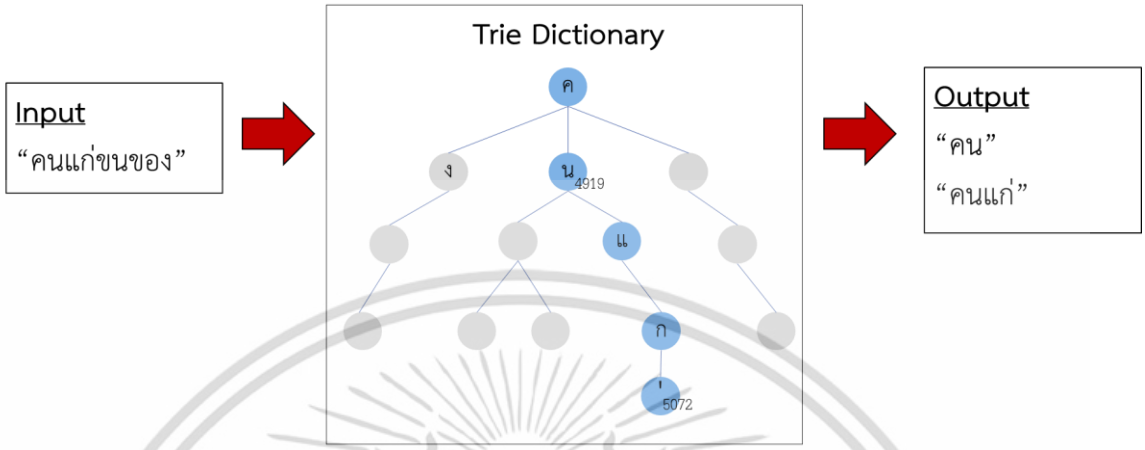


รูปที่ 3.20 แผนภาพการทำงานฟังก์ชัน findAllPrefix

3.3.4 พัฒนาฟังก์ชัน findAllPrefix

ฟังก์ชัน findAllPrefix คือ ฟังก์ชันที่ใช้ในการค้นหาคำขึ้นต้นของประโยคในบทความนั้นๆ หลังจากที่คิดความสัมพันธ์ของปัญหาและศึกษาค้นคว้าวิเคราะห์โครงสร้างข้อมูลแบบทรัย เพื่อพัฒนาให้มีหลักการทำงานตรงตามขอบเขตและตอบสนองความต้องการของผู้พัฒนา จึงคิดค้นฟังก์ชันนี้ขึ้นมาและทำการพัฒนาฟังก์ชัน findAllPrefix ลงในโครงสร้างข้อมูลแบบทรัย แสดงตัวอย่างกระบวนการทำงานได้ตามรูปในตัวอย่างที่ 3.21

ตัวอย่าง กระบวนการค้นหาคำขึ้นต้นของประโยค “คนแก่ขนของ”



รูปที่ 3.21 ตัวอย่างกระบวนการค้นหา Prefix

การพัฒนาฟังก์ชัน findAllPrefix โดยทำการเพิ่มลงไปในโครงสร้างข้อมูลแบบทรัยทั้ง 4 โครงสร้างจากข้อมูลข้างต้น แต่ละโครงสร้างข้อมูลแบบทรัยเป็นโครงสร้างที่มีหลักการทำงานแตกต่างกัน จึงจำเป็นต้องทำการพัฒนาฟังก์ชันที่มีรูปแบบการทำงานที่แตกต่างกันออกไปตามโครงสร้างข้อมูลแบบทรัยของแต่ละโครงสร้าง

ผู้พัฒนาได้พัฒนาฟังก์ชัน findAllPrefix ทั้งหมด 2 รูปแบบ คือ 1. ฟังก์ชัน findAllPrefix หลักการทำงานแบบลูป for 2. ฟังก์ชัน findAllPrefix หลักการทำงานแบบเวียนเกิด (Recursion) ซึ่งมีรายละเอียดดังนี้

1. ฟังก์ชัน findAllPrefix หลักการทำงานแบบลูป for

เป็นอัลกอริทึมที่ทำการพัฒนาลงใน TrieD ลำดับแรก ซึ่งมีหลักการทำงานแบบลูป for ทำให้ประสิทธิภาพแง่เวลาการทำงานของฟังก์ชันยังไม่ตรงตามขอบเขต ดังนั้น ฟังก์ชัน findAllPrefix จึงเป็นฟังก์ชันที่มีหลักการทำงานแบบ not Optimization ตัวอย่างการพัฒนาฟังก์ชัน findAllPrefix หลักการทำงานแบบลูป for ลงในโครงสร้าง TrieD แสดงได้ดังรูปที่ 3.22

```
Public static List<String> findAllPrefix (String str, TrieD<Integer> t) {
        List<String> list = new ArrayList<String> ();
        for (int I = 1; I <= str.length(); i++) {
                String word = str.subString (0, i);
                If (t.contains (word) == true) {
                        List. Add (word);
                }
        }
        return list;
}
```

รูปที่ 3.22 ฟังก์ชัน findAllPrefix หลักการทำงานแบบลูป for ของ TrieD

ในที่นี้ผู้พัฒนาได้ทำการพัฒนาฟังก์ชัน findAllPrefix โดยใช้หลักการทำงานแบบลูป for ลงใน TrieD แค่เพียง Trie เดียวเท่านั้น

2. ฟังก์ชัน findAllPrefix หลักการทำงานแบบเวียนเกิด (Recursion)

อัลกอริทึมการทำงานของฟังก์ชัน findAllPrefix ที่มีหลักการทำงานแบบเวียนเกิดทางผู้พัฒนาได้ทำการพัฒนาลงในโครงสร้างข้อมูลแบบทรัยทั้ง 4 โครงสร้าง โดยแต่ละโครงสร้างข้อมูลมีการทำงานในรูปแบบที่ต่างกัน จึงจำเป็นต้องทำการพัฒนาฟังก์ชันที่มีรูปแบบการทำงานที่แตกต่างกันออกไปตามโครงสร้างข้อมูลแบบทรัยของในแต่ละโครงสร้าง ดังรายละเอียดต่อไปนี้

- **TrieA**

ฟังก์ชัน findAllPrefix แบบ Optimization ของ TrieA มีโครงสร้างข้อมูลของ TrieNode ในรูปแบบการทำงานแบบลิ้งค์ลิสต์ (Linklist) แสดงอัลกอริทึมการทำงานดังรูปที่ 3.23

```
public List<String> findAllPrefix (String key) {
        List<String> result = new ArrayList<String> ();
        findAllPrefix ("", result, key, 0, root);
        return result;
}
public void findAllPrefix (String prefix, List<String> result, String word , int i,
        TrieNodeA array) {
                if (i >= word.length())
                        return;
                char ch = word.charAt(i);
                if (array.subNode(ch) == null)
                        return;
                else {
                   prefix = prefix + ch;
                   array = array.subNode(ch);
                   findAllPrefix (prefix, result, word, i+1, array);
                }
                if (array.isEnd == true)  result.add (prefix);
}
```

รูปที่ 3.23 ฟังก์ชัน findAllPrefix แบบ Optimization ของ TrieA

- **TrieB**

ฟังก์ชัน findAllPrefix แบบ Optimize ของ TrieB มีโครงสร้างข้อมูลของ TrieNode ในรูปแบบการทำงานแบบแฮชแม็พ (HashMap) แสดงอัลกอริทึมการทำงานดังรูปที่ 3.24

```
public List<String> findAllPrefix(String key) {
        List<String> result = new ArrayList<> (size);
        findAllPrefix (root, "", result, key, 0);
        return result;
}
public void findAllPrefix(TrieNode node, String prefix, final List<String> result,
    String key, int i) {
        if (i >= key.length()
                return;
        char ch = key.charAt (i);
        node = node.get(ch);
        prefix = prefix + ch;
        if (node == null)
                return;
        findAllPrefix (node, prefix, result, key, i+1);
        if (node.isWord()) {
                result.add (prefix);
        }
}
```

รูปที่ 3.24 ฟังก์ชัน findAllPrefix แบบ Optimization ของ TrieB

- **TrieC**

ฟังก์ชัน FindAllPrefix แบบ Optimize ของ TrieC มีโครงสร้างข้อมูลของ TrieNode ในรูปแบบเดียวกับ TrieB คือ การทำงานแบบแฮชแม็พ (HashMap) แตกต่างกันตรงที่ชื่อตัวแปรและฟังก์ชันการเรียกใช้งาน แสดงอัลกอริทึมการทำงานดังรูปที่ 3.25

```
public List<String> findAllPrefix(String key) {
        List<String> result = new ArrayList<String> ();
        findAllPrefix (root, "", result, key, 0);
        return result;
}
public void findAllPrefix(TrieNode node, String prefix, List<String> result,
     String key, int i) {
        if (i >= key.length())
            return;
        char ch = key.charAt(i);
         HashMap<Character, TrieNode> child = node.getChildren ();
        if (child.containsKey(ch))
        {       prefix += ch;
            node = child.get (ch);
            findAllPrefix (node, prefix, result, key, i+1);
            if ( node.isEnd() ){
                result.add (prefix);
            }
        }
        else  return;
}
```

รูปที่ 3.25 ฟังก์ชัน findAllPrefix แบบ Optimization ของ TrieC

- **TrieD**

ฟังก์ชัน findAllPrefix แบบ Optimize ของ TrieD มีโครงสร้างข้อมูลของ TrieNode ในรูปแบบการทำงานแบบ Node tree แสดงอัลกอริทึมการทำงานดังรูปที่ 3.26

```
public List<String> findAllPrefix(String pattern) {
        List<String> list = new ArrayList<String> ();
        findAllPrefix (root, new StringBuilder(), 0, pattern, list);
        Collections.reverse (list);
        return list;
}
public void findAllPrefix(Node<Value> x, StringBuilder prefix, int i, String pattern,
    List<String> list) {
        if (x == null)  return;
        char c = pattern.charAt (i);
        if (c < x.c)
                findAllPrefix (x.left, prefix, i, pattern, list);
        if (c == x.c) {
                if (i <= pattern.length() - 1 && x.val != null)
                        list.add(prefix.toString() + x.c);
                if (i < pattern.length() - 1) {
                        findAllPrefix (x.mid, prefix.append(x.c), i+1, pattern, list);
                        prefix.deleteCharAt(prefix.length() - 1);
                }
        }
        if (c > x.c)
                findAllPrefix(x.right, prefix, i, pattern, list);
}
```

รูปที่ 3.26 ฟังก์ชัน findAllPrefix แบบ Optimization ของ TrieD

3.3.5 การวิเคราะห์อัลกอริทึมฟังก์ชัน findAllPrefix

การวิเคราะห์อัลกอริทึม มีเป้าหมายเพื่อหาประสิทธิภาพของอัลกอริทึม นั่นคือ การประมาณค่าทรัพยากรที่จำเป็นต้องใช้ในการทำงาน เช่น เวลา หรือ หน่วยความจำ อัลกอริทึมส่วนใหญ่ถูกออกแบบมาเพื่อให้สามารถรองรับจำนวนอินพุทได้ไม่จำกัด ปกติแล้วประสิทธิภาพหรือความซับซ้อนของอัลกอริทึมจะวัดจากความสัมพันธ์ของจำนวนอินพุทกับเวลาที่ใช้ในการทำงานหรือพื้นที่หน่วยความจำที่ใช้

การพัฒนาฟังก์ชัน findAllPrefix ให้มีการทำงานที่มีประสิทธิภาพของอัลกอริทึมดีที่สุด ถูกต้องและรวดเร็วที่สุด จึงต้องวัดประสิทธิภาพเวลาในการทำงานของฟังก์ชัน โดยการพัฒนาฟังก์ชัน findAllPrefix ลงในโครงสร้างข้อมูลแบบทรัยในมีทั้งหมด 2 รูปแบบ คือ การใช้ for ในการวนลูป และการใช้เทคนิค Recursion เพื่อให้เห็นประสิทธิภาพเวลาการทำงานของอัลกอริทึมที่ชัดเจนมากขึ้น ผู้พัฒนาจึงทำการวิเคราะห์อัลกอริทึมของฟังก์ชันดังกล่าวให้ออกมาในรูปแบบของ Big-O เนื่องจากเป็นฟังก์ชั่นที่นิยมใช้มากที่สุด

3.3.6 พัฒนาฟังก์ชันการตัดคำภาษาไทย

การตัดคำภาษาไทยแบบคีย์เวิร์ดบนระบบการค้นหาข้อมูล เป็นการตัดคำภาษาไทยแบบพจนานุกรม ซึ่งแบ่งรูปแบบการตัดคำออกเป็น 2 รูปแบบ คือ

1. แบบ Safe segmentation
2. แบบ Unsafe segmentation

โดยการตัดคำภาษาไทยใช้หลักการทำงานแบบเวียนเกิดในการสร้างฟังก์ชัน segment เพื่อนำมาใช้สำหรับการตัดคำภาษาไทย จากขั้นตอนการดำเนินงานผู้พัฒนาได้นำคลังคำและโครงสร้างข้อมูลแบบทรัยที่เพิ่มฟังก์ชัน findAllPrefix มาทำงานร่วมกัน เพื่อให้ฟังก์ชัน segment มีการทำงานตรงตามวัตถุประสงค์ที่ผู้พัฒนาได้กำหนดไว้ ซึ่งหลักการทำงานแบบเวียนเกิดมีทั้งหมด 2 วิธี คือ

1. ฟังก์ชัน segment หลักการทำงานแบบค้นหาตามแนวลึก (Depth first search)
2. ฟังก์ชัน segment หลักการทำงานแบบค้นหาตามแนวกว้าง (Breadth first search)

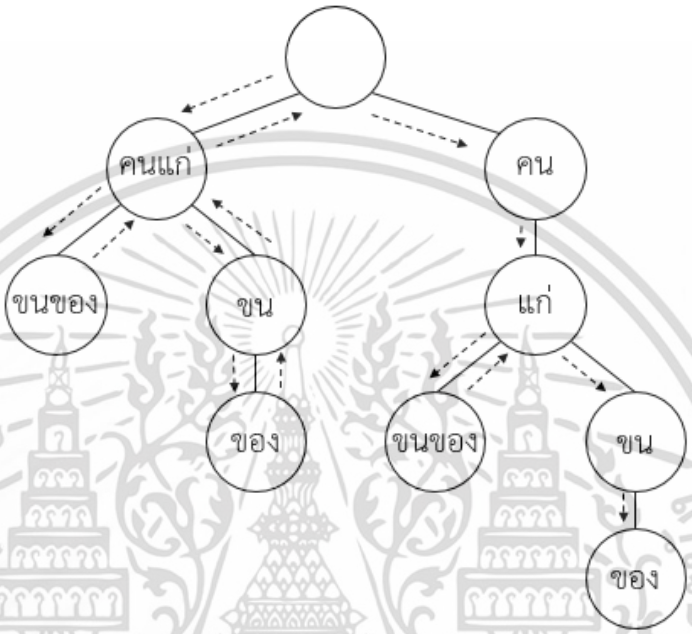
ซึ่งแต่ละวิธีมีรายละเอียดในการทำงาน ดังต่อไปนี้

1. ฟังก์ชัน segment หลักการทำงานแบบค้นหาตามแนวลึก

การทำงานแบบค้นหาตามแนวลึกเป็นการค้นหาที่เริ่มต้นจากโหนดราก (Root node) ที่อยู่บนสุด แล้วเดินลงมาให้ลึกที่สุด เมื่อถึงโหนดล่างสุด(Terminal node) ให้ย้อนขึ้นมาที่จุดสูงสุดของกิ่งเดียวกันที่มีกิ่งแยกและยังไม่ได้เดินผ่านแล้วเริ่มเดินลงจนถึงโหนดลึกสุดอีก ทำเช่นนี้สลับไปเรื่อยจนพบโหนดที่ต้องการหาหรือสำรวจครบทุกโหนดแล้ว ดังนั้น การค้นหาตามแนวลึกเมื่อนำมาใช้ในการพัฒนาฟังก์ชัน

การตัดคำภาษาไทยแบบเวียนเกิด จะให้ผลลัพธ์การตัดคำของทุกความเป็นไปได้ในแต่ละประโยคหรือบทความนั้น

<u>ตัวอย่าง</u> การทำงานแบบค้นหาตามแนวลึกของประโยค “คนแก่ขนของ” แสดงได้ดังรูปที่ 3.27



รูปที่ 3.27 การทำงานแบบค้นหาตามแนวลึกของประโยค “คนแก่ขนของ”

จากความสัมพันธ์ข้างต้นของปัญหานี้ การสร้างฟังก์ชัน segment โดยใช้หลักการทำงานแบบค้นหาตามแนวลึกของรูปแบบการตัดคำภาษาไทยทั้ง 2 รูปแบบ มีอัลกอริทึมการทำงาน ดังนี้

<u>กรณีที่ 1</u> แบบ Safe segmentation แสดงอัลกอริทึมการทำงานดังรูปที่ 3.28

```
1.  public List<List<String>> segment (String str) {
2.                  List<String> prefix = new ArrayList<String> ();
3.                  safe (prefix, str);
4.                  return segmentSolution;
5. }
6. private void safe (List<String> prefix, String str) {
7.          String remainStr;
8.          If (str.length () == 0) {
9.                  segmentSolution.add (new ArrayList<String> (prefix));
10.         }
11.         else {
12.                 List<String> dic = trie.findAllPrefix (str);
13.                 for (String word : dic) {
14.                         int check = str.indexOf(word);
15.                         if (check == 0) {
16.                                 prefix.add (word);
17.                                 remainStr = str.substring(word.length());
18.                                 safe (prefix, remainStr);
19.                                 prefix.remove (prefix.size() - 1);
20.                         }
21.                 }
22.         }
23. }
```

รูปที่ 3.28 ฟังก์ชัน segment หลักการทำงานแบบค้นหาตามแนวลึก กรณี Safe segmentation

จากรูปที่ 3.28 Segment() รับประโยคหรือบทความเข้ามา บรรทัดที่ 2 ประกาศ Prefix ขึ้นมาเพื่อจัดเก็บคำที่ได้จากการตัดคำของแต่ละประโยคหรือบทความนั้นๆ บรรทัดที่ 3 จะทำการเรียกใช้งาน safe() เริ่มจากฟังก์ชัน FindAllPrefix จะค้นหาคำขึ้นประโยคในแต่ละครั้งของการเรียกซ้ำ เงื่อนไข if ในบรรทัดที่ 8 จะทำการตรวจสอบประโยคหรือบทความที่รับเข้ามาว่ามีการตัดคำครบทั้งประโยคหรือบทความหรือไม่ ถ้าครบแล้วจะทำการจัดเก็บคำขึ้นต้นประโยคที่ได้มาจากการตัดคำจัดเก็บลงใน segmentSolution ถือเป็นเงื่อนไขเริ่มต้นของความสัมพันธ์ บรรทัดที่ 11-22 เป็นการทำงานแบบการเวียนเกิด บรรทัดที่ 13-21 เป็นวงวนของ Prefix ที่ได้มาในแต่ละครั้งจะนำ Prefix ที่ได้มาค้นหาในประโยคหรือบทความนั้น บรรทัดที่ 18 เป็นคำสั่งการเรียกซ้ำ โดยจะมีการเก็บ Prefix ที่ได้ แล้วตัด Prefix ของ

ประโยคหรือบทความนั้นออกเพื่อทำการค้นหา Prefix ในครั้งต่อไป ทำเช่นนี้ไปเรื่อยๆจนกระทั่งได้ผลลัพธ์ของทุกความเป็นไปได้ในแต่ละประโยคหรือบทความนั้น

กรณีที่ 2 แบบ Unsafe segment แสดงอัลกอริทึมการทำงานดังรูปที่ 3.29

```
1. public List<List<String>> segment(String str) {
2.         List<String> prefix = new ArrayList<String> ();
3.         unsafe (prefix, str);
4.         return segmentSolution;
5. }
6. private void unsafe(List<String> prefix, String str) {
7.         String remainStr;
8.         If (str.length () == 0) {
9.                   segmentSolution.add (new ArrayList<String> (prefix));
10.        }
11.        else {
12.                  List<String> dic = trie.findAllPrefix (str);
13.                  if (dic.isEmpty()) {
14.                            String skipChar = Character.toString (str.charAt(0));
15.                            prefix.add (skipChar);
16.                            remainStr = str.substring(1);
17.                            unsafe (prefix, remainStr);
18.                  }
19.                  for (String word : dic) {
20.                            int check = str.indexOf(word);
21.                            if (check == 0) {
22.                                      prefix.add (word);
23.                                      remainStr = str.substring(word.length());
24.                                      unsafe (prefix, remainStr);
25.                                      return;
26.                            }
27.                  }
28.        }
29. }
```

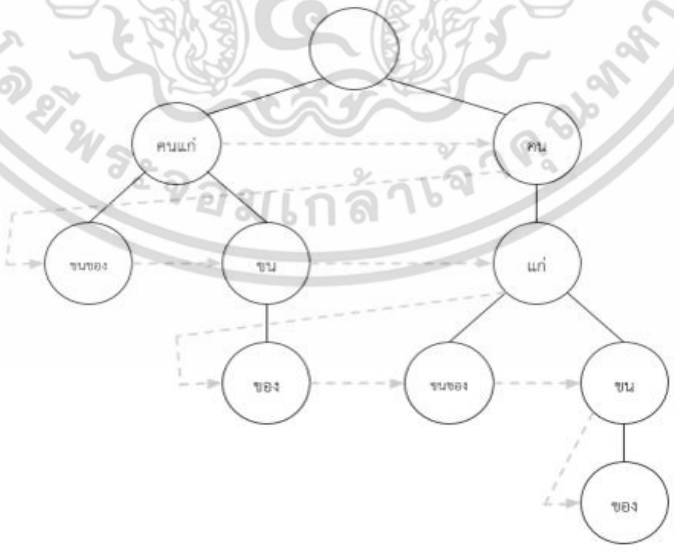
รูปที่ 3.29 ฟังก์ชัน segment หลักการทำงานแบบค้นหาตามแนวลึก กรณี Unsafe segmentation

จากรูปที่ 3.29 Segment() รับประโยคหรือบทความเข้ามา บรรทัดที่ 2 ประกาศ Prefix ขึ้นมาเพื่อจัดเก็บคำที่ได้จากการตัดคำของแต่ละประโยคหรือบทความนั้นๆ เช่นด้วยกับ unsafe() บรรทัดที่ 3 จะทำการเรียกใช้งาน unsafe() บรรทัดที่ 12 จะเรียกใช้ฟังก์ชัน findAllPrefix เพื่อค้นหาคำขึ้นประโยคในแต่ละครั้งของการเรียกซ้ำ เงื่อนไข if ในบรรทัดที่ 8 จะทำการตรวจสอบประโยคหรือบทความที่รับเข้ามาว่ามีการตัดคำครบทั้งประโยคหรือบทความหรือไม่ ถ้าครบแล้วจะทำการจัดเก็บคำขึ้นต้นประโยคที่ได้มาจากการตัดคำจัดเก็บลงใน segmentSolution บรรทัดที่ 11-28 เป็นการทำงานแบบการเรียกซ้ำ ซึ่ง unsafe() จะมีการทำงานที่แตกต่างจาก safe() ช่วงบรรทัดที่ 13-18 จะเป็นการตรวจสอบเมื่อประโยคหรือบทความไม่สามารถแทนด้วย Prefix ที่ได้มาหรือคำขึ้นต้นในประโยคนั้นๆไม่สามารถหา Prefix ได้ จึงทำการข้ามอักษรตัวนั้นด้วยการดึงอักษรตัวนั้นเก็บใน List แล้วทำการเรียกซ้ำต่อ จนกระทั่งจบประโยคหรือบทความนั้น

2. ฟังก์ชัน segment หลักการทำงานแบบค้นหาตามแนวกว้าง

การทำงานแบบค้นหาตามแนวกว้าง มีลักษณะการค้นหาลงไปทีละระดับ โดยการตรวจสอบระดับลูกทั้งหมดก่อนจึงลงไประดับถัดไป ซึ่งสามารถสร้างการค้นแบบนี้โดยใช้คิว (queue) เพื่อให้ปมถูกค้นตามลำดับของระดับชั้น ดังนั้น การค้นหาตามแนวกว้างเมื่อนำมาใช้ในการพัฒนาฟังก์ชันการตัดคำภาษาไทยแบบเวียนเกิด อัลกอริทึมการทำงานจึงนำคิวมาร่วมพัฒนา เพื่อให้ได้ผลลัพธ์การตัดคำของทุกความเป็นไปได้ในแต่ละประโยคหรือบทความนั้น

<u>ตัวอย่าง</u> การทำงานแบบค้นหาตามแนวกว้างของประโยค “คนแก่ขนของ” แสดงได้ดังรูปที่ 3.30



รูปที่ 3.30 การทำงานแบบค้นหาตามแนวกว้างของประโยค “คนแก่ขนของ”

แผนภาพการทำงานแบบค้นหาตามแนวกว้างของการตัดคำภาษาไทยที่มีการทำงานร่วมกับคิว จากแผนภาพคิวจะทำการรับข้อมูลเข้าสองตัว

- remainStr คือ ประโยคหรือบทความที่รับเข้ามา
- [ ] คือ List<String> ใช้ในการเก็บผลลัพธ์การตัดคำภาษาไทย

และการเรียกใช้ฟังก์ชัน dequeue จะมีการทำงานทั้งหมด 2 ขั้นตอนหลัก คือ

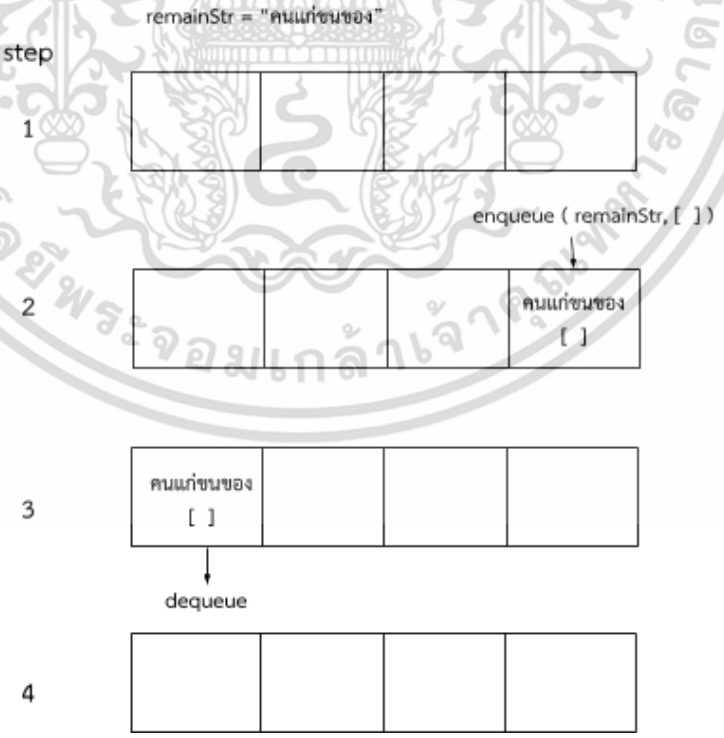
1. นำ findAllPrefix หาคำขึ้นต้นของประโยคหรือบทความนั้น
2. นำคำขึ้นต้นที่ได้จากการหา findAllPrefix มาใช้ในการตัดประโยค

หลังจาก 2 ขั้นตอนการทำงาน เมื่อทำการเรียกใช้ฟังก์ชัน enqueuer จะส่งประโยคหรือบทความที่ทำการตัดคำและคำขึ้นต้นที่เก็บในลิสต์ แสดงฟังก์ชัน dequeue ดังรูปที่ 3.31

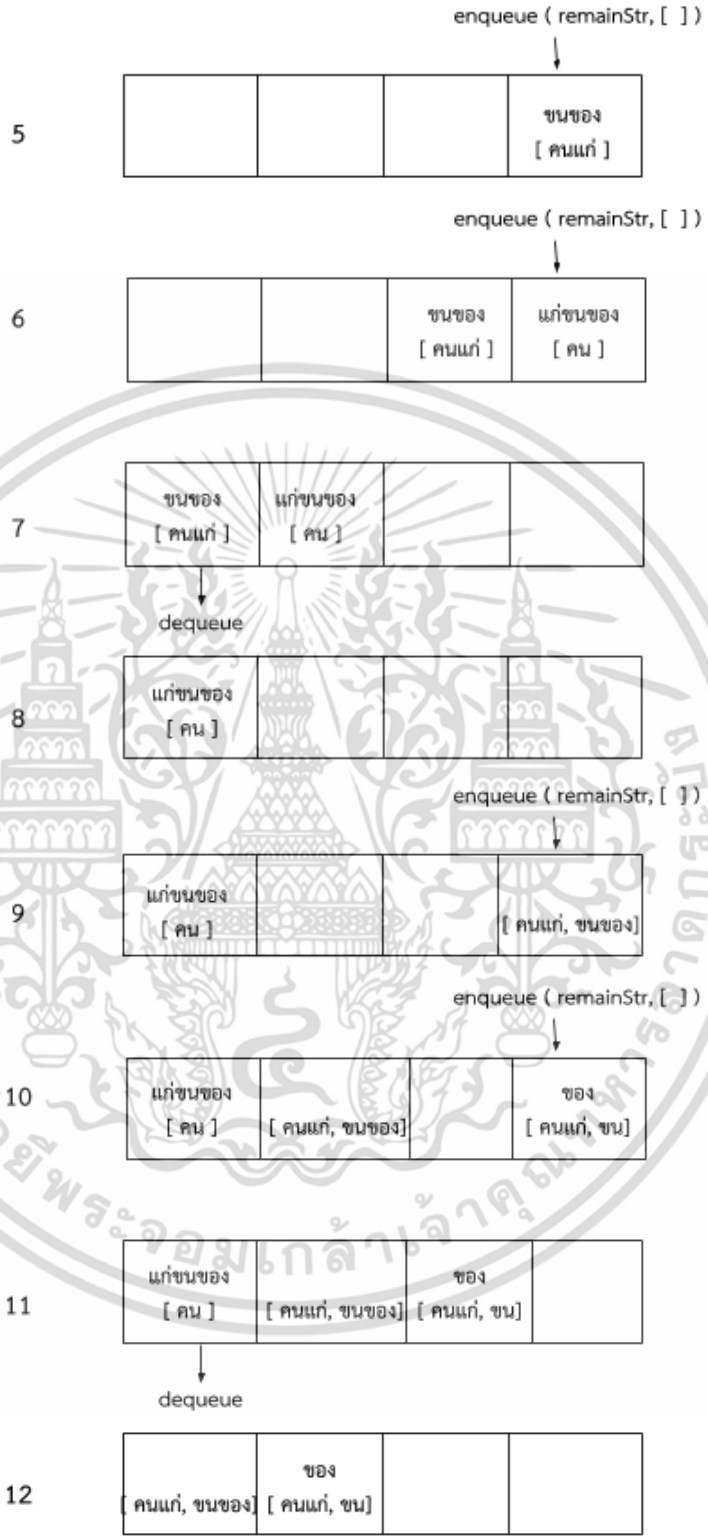
```
dequeue

    remainStr -> FindAllPrefix(remainStr);
    remainStr = remainStr.Substring(FindAllPrefix.length);
```

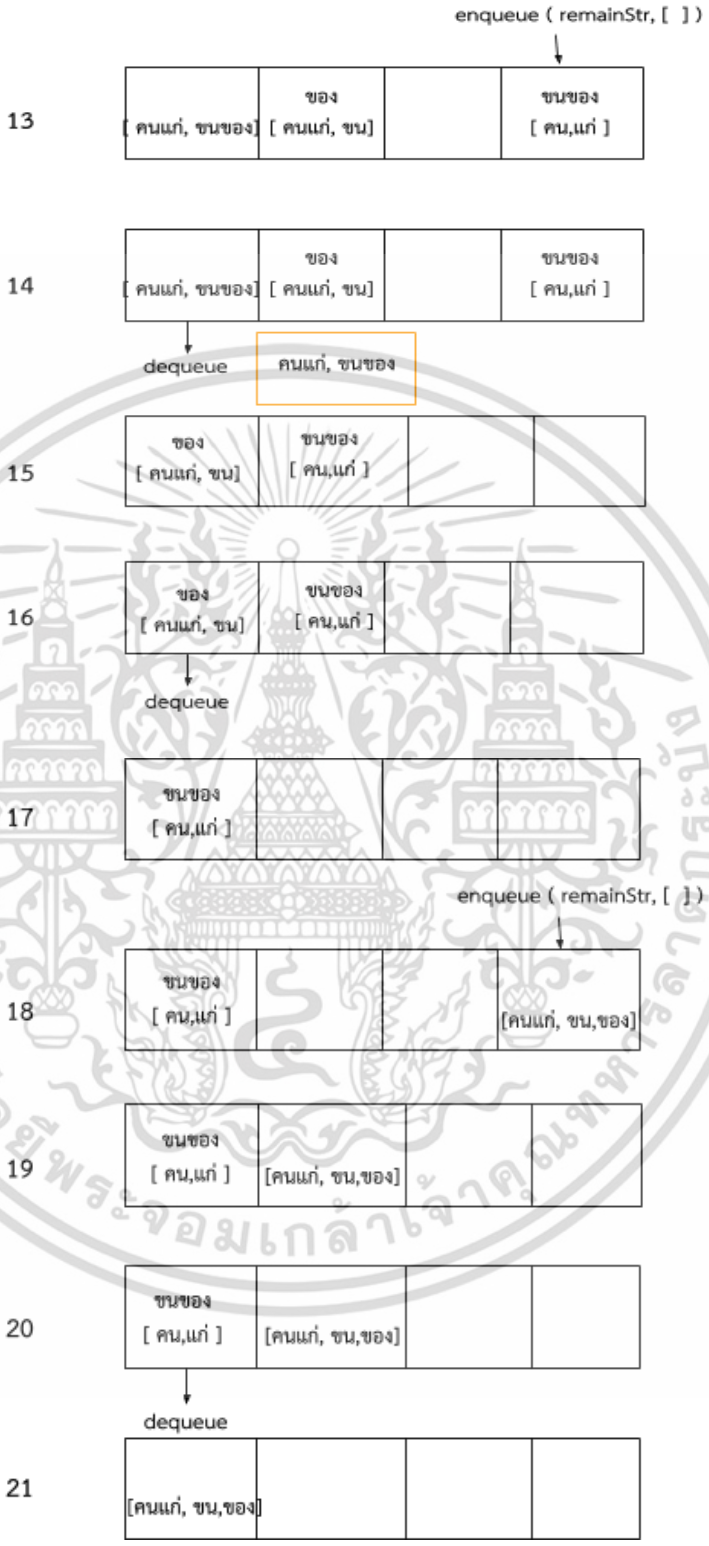
การตัดคำภาษาไทยของประโยค “คนแก่ขนของ” แสดงตัวอย่างกระบวนการทำงานดังแผนภาพที่ 3.31



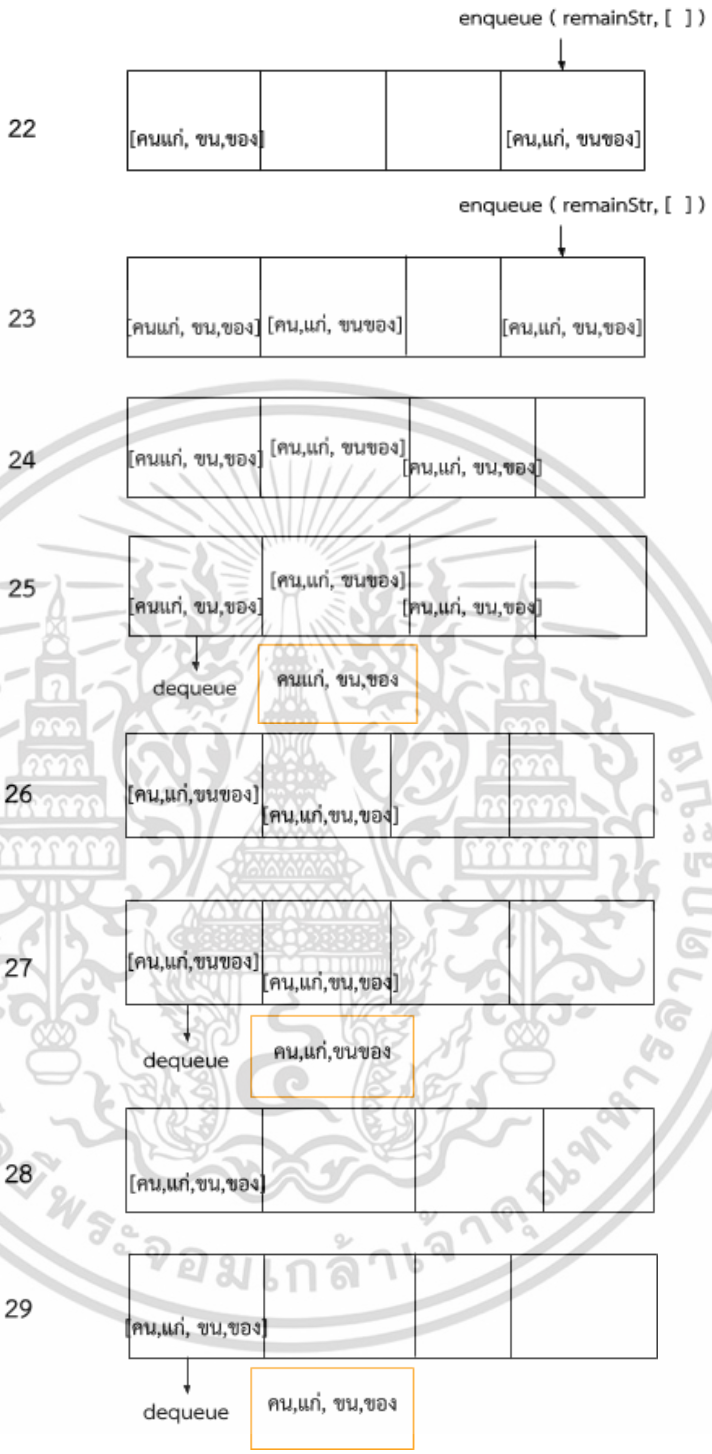
รูปที่ 3.31 ตัวอย่างกระบวนการทำงานการตัดคำภาษาไทยทำงานร่วมกับคิว (1)



รูปที่ 3.31 ตัวอย่างกระบวนการทำงานการตัดคำภาษาไทยทำงานร่วมกับคิว (2)



รูปที่ 3.31 ตัวอย่างกระบวนการทำงานการตัดคำภาษาไทยทำงานร่วมกับคิว (3)



รูปที่ 3.31 ตัวอย่างกระบวนการทำงานการตัดคำภาษาไทยทำงานร่วมกับคิว (4)

จากแผนภาพกระบวนการทำงานข้างต้น ทำให้ทั้งสองกรณีที่ถูกนำมาใช้ในการสร้างฟังก์ชัน segment โดยใช้หลักการทำงานแบบค้นหาตามแนวกว้าง ร่วมกับคิวนั้น ในแต่ละกรณีมีการทำงานดังต่อไปนี้

<u>กรณีที่ 1</u> แบบ Safe segmentation แสดงอัลกอริทึมการทำงานดังรูปที่ 3.32

```
1. public List<List<String>> segment(String str) {
2.         List<String> prefix = new ArrayList<String>();
3.         safe(prefix, str);
4.         return segmentSolution;
5. }
6. private void safe(List<String> prefix, String str) {
7.         List<String> solve = new ArrayList<String>();
8.         segment data = new segment(str, solve);
9.         Queue<segment> queue = new LinkedList<segment>();
10.        queue.add(data);
11.        while (!queue.isEmpty()) {
12.                segment current = queue.remove();
13.                List<String> dic = trie.findAllPrefix(current.remainStr);
14.                solve = new ArrayList<String>();
15.                String sentence = current.remainStr;
16.                solve.addAll(current.solution);
17.                for(String word : dic){
18.                        segment keep = new segment(word, solve);
19.                        keep.solution.add(word);
20.                        keep.remainStr = sentence.substring(word.length());
21.                        queue.add(keep);
22.                        if (keep.remainStr.length() == 0)
23.                                segmentSolution.add(keep.solution);
24.
25.                }
26.        }
27. }
```

รูปที่ 3.32 ฟังก์ชัน segment หลักการทำงานแบบค้นหาตามแนวลึก กรณี Safe segmentation

จากรูปที่ 3.32 segment() รับประโยคหรือบทความเข้ามา บรรทัดที่ 3 จะทำการเรียกใช้งาน safe() หลังจากนั้น safe() จะเริ่มทำงาน data ในบรรทัดที่ 8 จะประกอบด้วยประโยคหรือบทความที่รับเข้ามาและ List ใช้ในการเก็บคำที่ตัดได้จากประโยคหรือบทความนั้นๆ หลังจากนั้นจะทำการเพิ่ม data เข้าไปในคิว แล้วทำการตรวจสอบข้อมูลภายในคิวเพื่อนำมาตัดคำ เริ่มจากทำการหาคำขึ้นต้นของประโยคหรือบทความนั้นๆ เมื่อพบคำขึ้นต้นแล้วจะทำการตัดคำภาษาไทยเก็บใน List ส่วน remainStr จะเก็บประโยคหรือบทความที่ทำการตัดคำขึ้นต้นออกแล้ว ทำเช่นนี้ไปเรื่อยๆ จนกระทั่งจบประโยคหรือบทความนั้น

<u>กรณีที่ 2</u> แบบ Unsafe segment แสดงอัลกอริทึมการทำงานดังรูปที่ 3.33

```
1. public void segment (String str) {
2.         unsafe(str);
3. }
4. public void unsafe(String str) {
5.             List<String> solve = new ArrayList<String>();
6.             segment data = new segment(str, solve);
7.             Queue<segment> queue = new LinkedList<segment>();
8.             queue.add(data);
9.             while (!queue.isEmpty()) {
10.                         segment current = queue.remove();
11.                         List<String> dic = trie.findAllPrefix(current.remainStr);
12.                         solve = new ArrayList<String>();
13.                         String sentence = current.remainStr;
14.                         solve.addAll(current.solution);
15.                         if (!dic.isEmpty()) {
16.                                     for (String word : dic) {
17.                                                 segment keep = new segment(word, solve);
18.                                                 keep.solution.add (word);
19.                                                 keep.remainStr = sentence.substring(word.length());
20.                                                 queue.add (keep);
21.                                                 if (keep.remainStr.length() == 0)
22.                                                             return;
23.                                     }
24.                         } else {
25.                                     keep.solution.add (current.remainStr.charAt(0) + "");
26.                                     keep.remainStr = sentence.substring(1);
27.                                     queue.add (keep);
28.                                     if (keep.remainStr.length() == 0)
29.                                                 return;
30.                         }
31.             }
32. }
```

รูปที่ 3.33 ฟังก์ชัน segment หลักการทำงานแบบค้นหาตามแนวลึก กรณี Unsafe segmentation

จากรูปที่ 3.33 segment() มีหลักการทำงานที่คล้ายกับฟังก์ชัน segment ของ safe ใช้หลักการทำงานแบบการค้นหาแนวกว้าง เช่นเดียวกับกรณี safe segment แต่ในกรณีนี้จะทำการข้ามทีละอักขระเมื่อประโยคหรือบทความไม่สามารถแทนด้วยคำไทยได้ทั้งหมด ดังบรรทัดที่ 27-35 เมื่อประโยคหรือบทความไม่สามารถหา prefix ได้ ก็จะทำการข้ามไปทีละอักขระ เก็บไว้ใน solution ส่วน remainStr จะทำการตัดออกทีละอักขระเช่นกัน ทำเช่นนี้ไปเรื่อยๆ จนกระทั่งจบประโยคหรือบทความนั้นๆ

3.3.7 โมดูลการตัดคำ (Analyzer)

โมดูลการตัดคำเป็นคลาสบนจาวาที่ถูกสร้างขึ้นเพื่อใช้ในการตัดคำภาษาไทยแบบพจนานุกรมโดยเก็บข้อมูลในรูปแบบสายอักขระของโครงสร้างทรัย

<u>ส่วนประกอบของ</u> โมดูลการตัดคำ

- Deserialize คือ การแปลงสายอักขระให้เป็นโครงสร้างข้อมูลเชิงวัตถุ 2 ไลบรารี่ คือ
  - Protostuff
  - Java serialize
- โครงสร้างข้อมูลแบบทรัยที่พัฒนาฟังก์ชัน findAllPrefix 2 โครงสร้าง คือ
  - TrieA
  - TrieB
- รูปแบบการตัดคำภาษาไทยแบบพจนานุกรม มี 2 รูปแบบ คือ
  - **การตัดคำแบบปลอดภัย** (Safe segmentation) คือ วิธีการตัดคำที่ให้ผลลัพธ์เป็นทุกคำของการตัดคำ ซึ่งมีการทำงานแบบย้อนรอยและการตัดคำโดยให้ผลลัพธ์ของทุกความเป็นไปได้ในแต่ละประโยคหรือบทความนั้น
  - **การตัดคำแบบไม่ปลอดภัย** (Unsafe segmentation) คือ การตัดคำในกรณีที่พบประโยคหรือบทความที่มีตัวสะกดผิด ซึ่งอาจเกิดจากการพิมพ์ผิดหรือภายในคลังคำไม่มีคำใหม่

โดยสร้างโมดูลการตัดคำขึ้นมาเพื่อนำมาใช้ในการตัดคำภาษาไทยบนจาวามีทั้งหมด 3 โมดูล ดังนี้

1. ProtostuffSafeAnalyzer

ประกอบด้วย

- Protostuff
- TrieA
- Safe segment

2. ProtostuffUnsafeAnalyzer

ประกอบด้วย

- Protostuff
- TrieA
- Unsafe segment

3. JavaSerializeUnsafeAnalyzer

ประกอบด้วย

- Java serialize
- TrieB
- Unsafe segment

ในการสร้างโมดูลการตัดคำ โดยสร้างประโยคทดสอบสองรูปแบบ คือ ประโยคหรือบทความที่สามารถแทนด้วยคำภาษาไทยได้ทั้งหมดและประโยคหรือบทความที่ไม่สามารถแทนด้วยคำภาษาไทยได้ทั้งหมด เพื่อให้เห็นความแตกต่างระหว่างการตัดคำ โดยใช้คำที่มีอยู่ในคลังคำ ดังนี้

- **ประโยคที่แทนด้วยคำภาษาไทยได้ทั้งหมด (Clean-sentence)**

การสร้างรูปแบบประโยค จะสุ่มคำจากคลังคำมาเรียงต่อกันเป็นประโยคโดยไม่สนใจความหมายของประโยคนั้นๆ ดังประโยคตัวอย่างในรูปที่ 3.34

บัตรห้องสมุดติติงพิซซ่าฮัทซัลเฟตเพ่งดูร้องรำทำเพลงบ่าวสาวถ่อมตัวเทศบาลเมืองความชื่นชม
หวันยิหวาผู้จ้างสำเนาสับสนปฏิกิริยาเคมีกุลาตีไม้อเจลกอินทรชิตมหาชัย

รูปที่ 3.34 ตัวอย่างประโยค Clean-sentence

- **ประโยคที่ไม่สามารถแทนด้วยคำภาษาไทยได้ทั้งหมด (Dirty-sentence)**

การสร้างรูปแบบประโยค จะสุ่มตัวอักษรภาษาอังกฤษแทรกไปในประโยคที่ทำการสุ่มมาเรียงต่อกัน โดยการสุ่มตัวอักษรภาษาอังกฤษจะช่วยให้ไม่เกิดคำที่ซ้ำซ้อนในภาษาไทย ดังตัวอย่างที่แสดงในรูปที่ 3.35

บัตรห้องสมุHดติติงพิซซ่าฮัทซัลเฟตเพ่งดูร้องรำทำเพลงบ่าวสาวถ่อมตัวเทศบาลเมืองความชื่นชม
ตั้งใจฟังเหล่านี้กราวรูดซวนความมักง่ายแกนนำหEลับตาวอลล์ครหาหัก

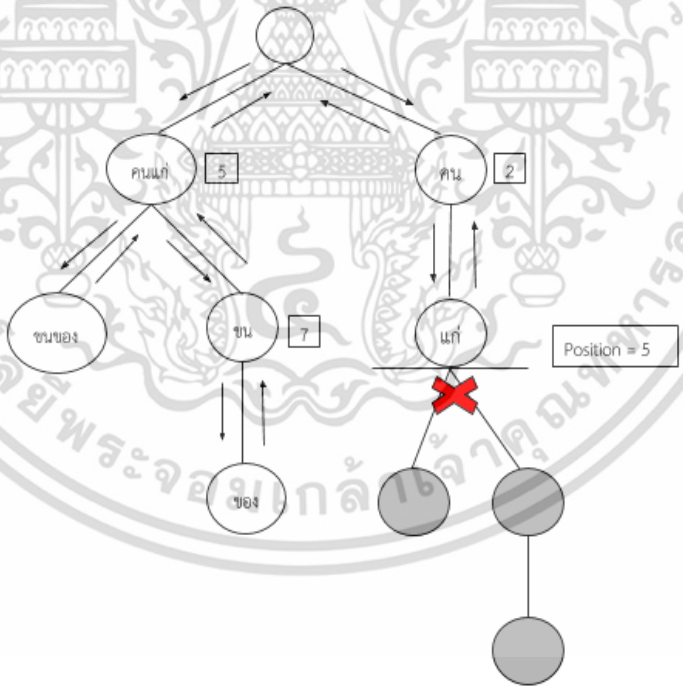
รูปที่ 3.35 ตัวอย่างประโยค Dirty-sentence

3.3.8 พัฒนาฟังก์ชันการตัดคำภาษาไทยโดยเทคนิคการขยายและจำกัดเขต (Branch and bround)

เป็นเทคนิคที่นำมาใช้ในการเพิ่มประสิทธิภาพของการตัดคำภาษาไทย ด้วยหลักการแบ่งหรือแตกกิ่งและการตัดหรือการหยุด โดยเริ่มต้นจากปัญหาที่มีขนาดใหญ่และแบ่งเป็น ปัญหาย่อยๆ จากนั้นพิจารณาขอบเขต ของคำตอบสำหรับปัญหาย่อย และพิจารณาตัดปัญหาที่ไม่สามารถให้คำตอบที่ดีที่สุดในขณะนั้นได้ และทำซ้ำกับทุกปัญหาย่อยๆ จนกระทั่งพบปัญหาย่อยที่ให้คำตอบที่ดีที่สุด

โดยขอบเขตของการหยุดการทำงานในการตัดคำภาษาไทย ผู้พัฒนาได้ศึกษาเกี่ยวกับ backtracking index ที่จะนำมาใช้ในการเพิ่มประสิทธิภาพของการตัดคำภาษาไทย ซึ่งการนำ backtracking index เข้ามาใช้ในการเพิ่มประสิทธิภาพของการตัดคำภาษาไทยนี้ จะช่วยเพิ่มความเร็วในการตัดคำและผลลัพธ์ที่ทุกความเป็นได้ของรูปแบบคำ

ตัวอย่าง การตัดคำภาษาไทยที่เพิ่มประสิทธิภาพด้วยเทคนิคการขยายและจำกัดเขต ของประโยค “คนแก่ขนของ” แสดงดังรูปที่ 3.36



รูปที่ 3.36 การทำงานแบบการขยายและจำกัดเขตของประโยค “คนแก่ขนของ”

จากการทำงานของรูปที่ 3.36 ทำให้ถูกนำมาใช้ในการสร้างฟังก์ชัน segment โดยใช้หลักการทำงานแบบค้นหาแนวลึกเพิ่มประสิทธิภาพของการตัดคำภาษาไทยด้วยเทคนิคการขยายและจำกัดเขต มีอัลกอริทึมการทำงาน แสดงดังรูปที่ 3.37

```
1. public List<List<String>> segment(String str) {
2.         List<String> prefix = new ArrayList<String>();
3.         List<Integer> index = new ArrayList<Integer>();
4.         safe(0, index, prefix, str);
5.         return segmentSolution;
6. } private void safe(int n, List<Integer> index, List<String> prefix, String str) {
7.         List<String> dic = trie.findAllPrefix(str);
8.         String remainStr;
9.         if(str.length() == 0) segmentSolution.add(new ArrayList<String>(prefix));
10.        if(segmentSolution.isEmpty() && index.contains(n)) return;
11.        if(!segmentSolution.isEmpty() && index.contains(n)) {
12.                segmentSolution.add(new ArrayList<String>(prefix));
13.                return;
14.        }
15.        for(String word : dic) {
16.                int check = str.indexOf(word);
17.                if(check == 0){
18.                        n = n + word.length();
19.                        prefix.add(word);
20.                        remainStr = str.substring((word).length());
21.                        safe(n, index, prefix, remainStr);
22.                        prefix.remove(prefix.size() - 1);
23.                        if(!index.contains(n))
24.                                index.add(n);
25.                        n = n - word.length();
26.                }
27.        }
28. }
```

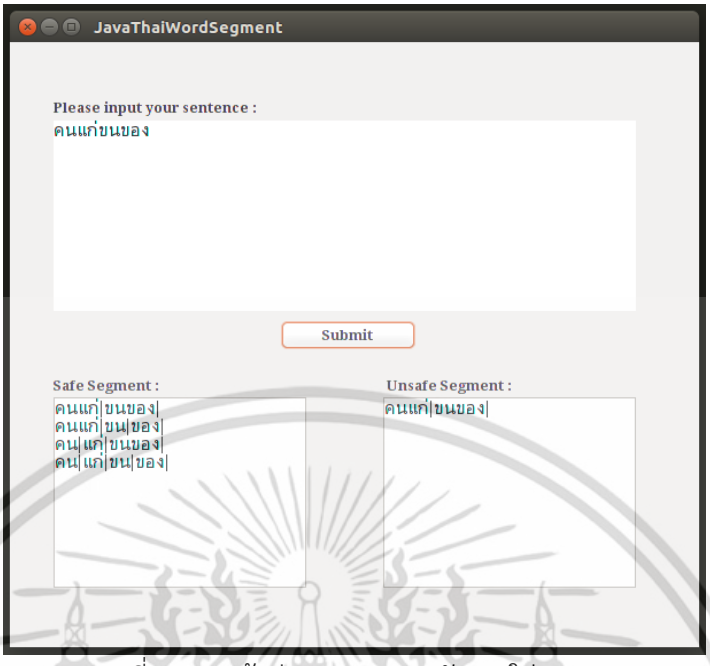
รูปที่ 3.37 ฟังก์ชัน segment หลักการทำงานแบบค้นหาแนวลึกเพิ่มประสิทธิภาพของการตัดคำภาษาไทยด้วยเทคนิคการขยายและจำกัดเขต

จากรูปที่ 3.37 ในส่วนของการทำงานจะคล้ายคลึงกับการค้นหาแนวลึกกรณี safe segment จะต่างกันที่บรรทัดที่ 9-14 จะเป็นการตรวจสอบการทำงานในการตัดคำภาษาไทย บรรทัดที่ 23-24 จะเป็นการเก็บตำแหน่งของแต่ละคำในประโยคที่ยังไม่มีอยู่ใน index บรรทัดที่ 11 ทำการตรวจสอบว่ามีผลลัพธ์การตัดคำและตำแหน่งตัวอักษรมีใน index หรือไม่ ถ้าตรงตามเงื่อนไขจะทำการย้อนกลับไปตัดยังประโยครูปแบบถัดไป กรณีนี้ใช้ในกรณีที่ประโยคหรือบทความไม่สามารถแทนด้วยคำไทยได้ทั้งหมด บรรทัดที่ 10-14 จะทำการตรวจสอบเช่นเดียวกับบรรทัดที่ 11 แต่ถ้าตรงตามเงื่อนไขจะทำการเก็บผลลัพธ์ที่ได้และย้อนกลับไปตัดยังประโยครูปแบบถัดไป กรณีนี้ใช้ในกรณีที่ประโยคหรือบทความสามารถแทนด้วยคำไทยได้ทั้งหมด แล้วทำการเรียกซ้ำต่อ จนกระทั่งจบประโยคหรือบทความนั้น

3.3.9 Demo

ในการทำ Demo เพื่อเป็นการแสดงให้เห็นผลลัพธ์ของการทำงานในการตัดคำภาษาไทยให้เห็นภาพลักษณ์ที่ชัดเจนมากยิ่งขึ้น โดยแสดงรูปแบบหน้าต่างของ Demo ดังรูปที่ 3.38 และ 3.39

รูปที่ 3.38 หน้าต่างแสดง UI ก่อนใส่ input



รูปที่ 3.39 หน้าต่างแสดง UI หลังจากใส่ input

บทที่ 4
ผลการดำเนินงาน

ผลการปฏิบัติงานสหกิจศึกษาพัฒนาโครงงาน การตัดคำภาษาไทยแบบคีย์เวิร์ดบนระบบการค้นหาข้อมูล (Thai word segment keyword on search engine) เป็นการตัดคำแบบการใช้พจนานุกรม (dictionary base) โดยใช้โครงสร้างข้อมูลแบบทรัย (Trie Structure) มาเป็นโครงสร้างข้อมูลหลักที่ใช้ในการสืบค้นคำ และมีการทดสอบเพื่อวัดประสิทธิภาพแง่เวลาการทำงาน ซึ่งแบ่งออกเป็น

- วัดประสิทธิภาพการทำงานโครงสร้างข้อมูลแบบทรัยของแต่ละโครงสร้าง
- วัดประสิทธิภาพการทำงานการแปลงโครงสร้างข้อมูลเชิงวัตถุให้เป็นสายอักขระบนภาษาจาวาของแต่ละไลบารี ในแต่ละโครงสร้างข้อมูลแบบทรัยที่มีคลังคำอยู่ภายในโครงสร้างข้อมูลแบบทรัยทั้งหมด 40,000 คำ
- วัดประสิทธิภาพการทำงานโปรแกรมการตัดคำภาษาไทยของแต่ละเทคนิคที่นำมาใช้ในการพัฒนา
- วัดประสิทธิภาพการทำงานของโปรแกรมการตัดคำภาษาไทยที่ทำการเพิ่มประสิทธิภาพโดยใช้เทคนิคการขยายและจำกัดเขต (branch and bound)

โดยแสดงผลการดำเนินงานของโครงงาน การตัดคำภาษาไทยแบบคีย์เวิร์ดบนระบบการค้นหาข้อมูล ได้ดังนี้

4.1 การวิเคราะห์อัลกอริทึม

ในการพัฒนาฟังก์ชัน findAllPrefix เพิ่มในโครงสร้างข้อมูลแบบทรัย ได้ทำการวัดประสิทธิภาพเวลาการทำงานของอัลกอริทึมฟังก์ชัน findAllPrefix ที่ทำการเพิ่มลงไปในทรัยทั้ง 4 โครงสร้าง แสดงให้เห็นว่าไม่สามารถวัดประสิทธิภาพการทำงานของฟังก์ชันด้วยวิธีการวิเคราะห์อัลกอริทึมได้ ด้วยเหตุผลดังนี้

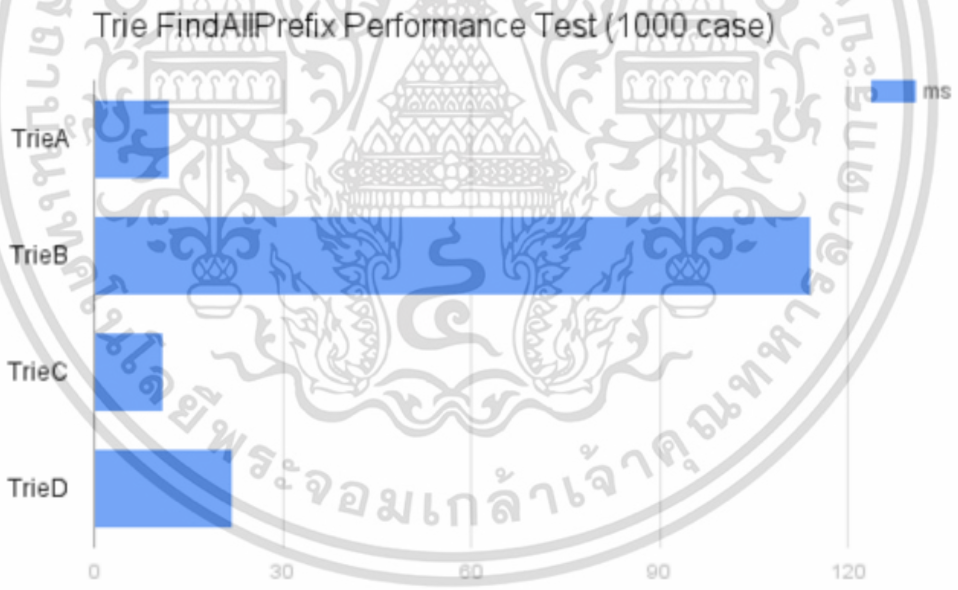
- รายการคำที่ใช้นำไปสร้างเกิดโครงสร้างไร้ความเป็นระเบียบในทรัย ทำให้ไม่สามารถวิเคราะห์อัลกอริทึมของฟังก์ชัน findAllPrefix ให้ออกมาในรูปแบบของ Big-O ได้ เนื่องจากแสดงเวลาการทำงานของฟังก์ชันได้อย่างไม่ชัดเจน
- โครงสร้างข้อมูลแบบทรัยที่ทำการค้นคว้าและรวบรวมมาใช้ในทดลองในโครงงานนี้ มีโครงสร้างข้อมูลที่แตกต่างกันในแต่ละโครงสร้างจากรายการคำที่เหมือนกัน
- ขอบเขตของโครงงานจะใช้แบบรายการคำรูปแบบเดียวกันสำหรับแต่ละโครงสร้างข้อมูลแบบทรัยเท่านั้น จะไม่ครอบคลุมถึงการหาลำดับหรือการสลับรายการคำของรายการ ที่ทำให้ประสิทธิภาพของโครงสร้างข้อมูลแบบทรัยออกมาสูงที่สุด

จึงทำการวัดประสิทธิภาพโครงสร้างข้อมูลแบบทรัยที่ทำการเพิ่มฟังก์ชัน findAllPrefix ด้วยวิธีการวัดประสิทธิภาพเวลาในการทำงานของฟังก์ชัน findAllPrefix แทนการวัดประสิทธิภาพการทำงานของฟังก์ชันด้วยหลักการวิเคราะห์อัลกอริทึม แสดงได้ดังหัวข้อถัดไป

4.2 การวัดประสิทธิภาพการทำงานของโครงสร้างข้อมูลแบบทรัย

การวัดประสิทธิภาพการทำงานของโครงสร้างข้อมูลแบบทรัย เป็นการทดสอบประสิทธิภาพการทำงานของโครงสร้างข้อมูลแบบทรัยที่ทำการพัฒนาฟังก์ชัน findAllPrefix ลงไปในโครงสร้างแล้ว เพื่อแสดงให้เห็นถึงเวลาการทำงานและบ่งบอกข้อแตกต่างของการทำงานในแต่ละโครงสร้าง

การวัดประสิทธิภาพการทำงานของโครงสร้างข้อมูลแบบทรัยจะทำการสุ่มคำที่มีอยู่ในคลังคำมาเรียงต่อกันเป็นประโยคที่มีความยาวประมาณ 150 ตัวอักษร ซึ่งมีทั้งหมด 1000 ประโยค โดยนำฟังก์ชัน findAllPrefix ของแต่ละโครงสร้างข้อมูลแบบทรัยมาใช้ในการทดสอบ แสดงผลการวัดประสิทธิภาพเวลาการทำงาน ดังกราฟในรูปที่ 4.1



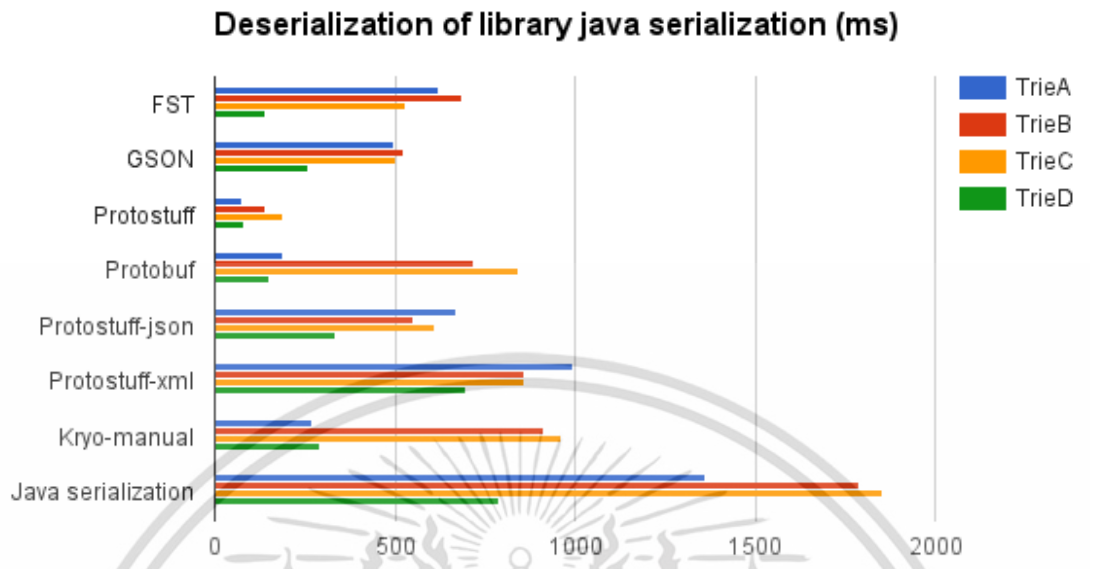
รูปที่ 4.1 กราฟแสดงประสิทธิภาพเวลาการทำงานของโครงสร้างข้อมูลแบบทรัยที่ทำการพัฒนาฟังก์ชัน findAllPrefix

จากกราฟในรูปที่ 4.1 แสดงให้เห็นว่า โครงสร้างข้อมูลแบบทรัยที่ทำการพัฒนาฟังก์ชัน findAllPrefix ของ TrieA และ TrieC เป็นโครงสร้างข้อมูลที่มีเวลาการทำงานใกล้เคียงกันและใช้เวลาในการทำงานที่ค่อนข้างน้อย และ TrieB เป็นโครงสร้างข้อมูลที่ใช้เวลาการทำงานมากที่สุด

4.3 การวัดประสิทธิภาพการทำงานของการแปลงโครงสร้างข้อมูลเชิงวัตถุให้เป็นสายอักขระ

การวัดประสิทธิภาพการทำงานของการแปลงโครงสร้างข้อมูลเชิงวัตถุให้เป็นสายอักขระ เป็นการทดสอบประสิทธิภาพเวลาการทำงานของแต่ละไลบารี่ ในแต่ละโครงสร้างข้อมูลแบบทรัย โดยจะทำการวัดประสิทธิภาพเวลาการทำงานของการแปลงสายอักขระให้เป็นโครงสร้างข้อมูลเชิงวัตถุ (deserialization) เพื่อเลือกไลบรารีที่ทำงานร่วมกับโครงสร้างข้อมูลแบบทรัยที่มีคลังคำอยู่ภายในโครงสร้าง แล้วให้ประสิทธิภาพเวลาการทำงานที่รวดเร็วที่สุดเมื่อมีการเรียกใช้โปรแกรมตัดคำภาษาไทย

การแปลงโครงสร้างข้อมูลแบบทรัยที่มีคลังคำอยู่ภายในโครงสร้างทั้งหมด 40,000 คำ ให้เป็นสายอักขระ (Serialization) แล้วนำไปใช้ในการตัดคำภาษาไทยแบบคีย์เวิร์ดบนระบบการค้นหาข้อมูล เพื่อประหยัดเวลาในการเพิ่มคลังคำเข้าไปในโครงสร้างข้อมูลแบบทรัยทุกครั้งที่ทำการตัดคำภาษาไทย โดยทางผู้พัฒนาเน้นประสิทธิภาพเวลาการทำงานที่รวดเร็วของการตัดคำภาษาไทย ผู้พัฒนาจึงให้ความสำคัญกับการแปลงสายอักขระให้เป็นโครงสร้างข้อมูลเชิงวัตถุ ในแต่ละโครงสร้างข้อมูลแบบทรัย เพราะในกระบวนการตัดคำจำเป็นต้องเพิ่มคลังคำเข้าไปในโครงสร้างข้อมูลแบบทรัยในทุกๆครั้งของการตัดคำ ผู้พัฒนาจึงใช้หลักการทำงานของการแปลงสายอักขระให้เป็นโครงสร้างข้อมูลเชิงวัตถุนำไปใช้ในการตัดคำภาษาไทย เพื่อความสะดวกและความรวดเร็วในการการตัดคำภาษาไทย แสดงผลการวัดประสิทธิภาพเวลาการทำงานของการแปลงโครงสร้างข้อมูลเชิงวัตถุให้เป็นสายอักขระ ดังกราฟในรูปที่ 4.2



รูปที่ 4.2 กราฟแสดงประสิทธิภาพเวลาการทำงานของ Deserialization

จากกราฟรูปที่ 4.2 แสดงให้เห็นว่า Protostuff เป็นการแปลงโครงสร้างข้อมูลเชิงวัตถุให้เป็นสายอักขระบนภาษาจาวาของแต่ละไลบารี (Java serializable library) ที่มีประสิทธิภาพเวลาการทำงานรวดเร็วที่สุดในทุกๆโครงสร้างข้อมูลแบบทรัย และ Java serialization เป็นไลบารี่ที่มีประสิทธิภาพเวลาการทำงานที่ช้าที่สุดของแต่ละโครงสร้างข้อมูลแบบทรัย

4.4 การวัดประสิทธิภาพการทำงานของโมดูลการตัดคำ

การวัดประสิทธิภาพการทำงานของโมดูลการตัดคำ เป็นการทดสอบประสิทธิภาพเวลาการทำงานที่มีความรวดเร็วที่สุดของการตัดคำโดยใช้เทคนิคการค้นหาที่เร็วที่สุดจากการศึกษาและนำการวัดประสิทธิภาพก่อนหน้านี้มาใช้เป็นส่วนหนึ่งในการตัดคำภาษาไทย โดยข้อมูลที่จะนำมาใช้ในการทดสอบประสิทธิภาพเวลาการทำงานของโมดูลการตัดคำ มี 2 รูปแบบ ได้แก่

รูปแบบที่ 1 ทำการสุ่มคำทั้งหมดในคลังคำ 10 คำมาเรียงต่อกันเป็นประโยคทั้งหมด 1000 ประโยค แทนด้วยคำว่า Clean - sentence

รูปแบบที่ 2 ทำการสุ่มคำทั้งหมดในคลังคำ 10 คำ มาเรียงต่อกันเป็นประโยคทั้งหมด 1000 ประโยค และสุ่มใส่อักขระพิเศษลงไปในประโยคละ 1 อักขระ แทนด้วยคำว่า Dirty - sentence

การวัดประสิทธิภาพการทำงานของโมดูลการตัดคำ แบ่งออกเป็น 2 กรณี

<u>กรณีที่ 1</u> การวัดประสิทธิภาพการทำงานของโมดูลการตัดคำ ของรูปแบบเทคนิคที่ใช้ในการตัดคำภาษาไทย คือ safe segmentation และ unsafe segmentation โดยนำการแปลงสายอักขระให้เป็น

โครงสร้างข้อมูลเชิงวัตถุที่มีเวลาในการทำงานเร็วที่สุด ในที่นี้คือ Protostuff และโครงสร้างข้อมูลแบบทรัย คือ TrieA มาใช้ในการสร้าง Analyzer ซึ่งมีทั้งหมด 2 Analyzer ได้แก่

- **ProtostuffSafeAnalyzer**

  ประกอบด้วย

  Protostuff + TrieA + safe segmentation

- **ProtostuffUnsafeAnalyzer**

  ประกอบด้วย

  Protostuff + TrieA + unsafe segmentation

โดยใช้ clean-sentence และ dirty-sentence เป็นข้อมูลที่จะนำมาใช้ในการวัดประสิทธิภาพเวลาการทำงานของโมดูลการตัดคำ แสดงผลข้อมูลได้ดังตารางที่ 4.1

ตารางที่ 4.1 แสดงประสิทธิเวลาการทำงานของ Analyzer ระหว่าง safe segment และ unsafe segment

	ProtostuffSafeAnalyzer	ProtostuffUnsafeAnalyzer
Clean - sentence	15321 ms	21 ms
Dirty - sentence	345 ms	18 ms

จากตารางที่ 4.1 การตัดคำภาษาไทยของ clean-sentence โดยใช้ ProtostuffSafeAnalyzer จะมีเวลาในการทำงานที่มากกว่า การใช้ ProtostuffUnsafeAnalyzer ค่อนข้างมาก เนื่องจากการตัดคำภาษาไทยแบบ safe segmentation จะเป็นการตัดคำภาษาไทยที่สามารถแสดงทุกผลลัพธ์ของความเป็นไปได้ จึงทำให้ใช้เวลาในการทำงานที่ค่อนข้างมาก แต่ทางผลลัพธ์จะทำให้ได้ทุกคำที่เป็นไปได้ภายในประโยค ซึ่งแตกต่างจาก unsafe segmentation ที่จะทำการตัดคำที่แสดงผลลัพธ์เพียงหนึ่งผลลัพธ์เท่านั้นจึงทำให้ใช้เวลาในการตัดคำที่เร็วกว่า safe segmentation มากพอสมควร

<u>กรณีที่ 2</u> การวัดประสิทธิภาพการทำงานของโมดูลการตัดคำภาษาไทยในรูปแบบ unsafe segmentation โดยใช้การแปลงสายอักขระให้เป็นโครงสร้างข้อมูลเชิงวัตถุที่มีเวลาในการทำงานเร็วที่สุดและการทำงานที่ช้าที่สุด ในที่นี้คือ Protostuff และ Java serialize โครงสร้างข้อมูลแบบทรัยที่นำมาใช้คือ TrieA และ TrieB มาใช้ในการสร้าง Analyzer ซึ่งมีทั้งหมด 2 Analyzer ได้แก่

- **ProtostuffUnsafeAnalyzer**

  ประกอบด้วย

  Protostuff + TrieA + unsafe segment

- **JavaSerializeUnsafeAnalyzer**

  ประกอบด้วย

  Java serialize + TrieB + unsafe segment

โดยนำ clean-sentence และ dirty-sentence เป็นข้อมูลที่จะนำมาใช้ในการวัดประสิทธิภาพการทำงานของ Analyzer แสดงผลข้อมูลได้ดังตาราง 4.2

ตารางที่ 4.2 แสดงประสิทธิเวลาการทำงานของ Analyzer ในกรณีที่ดีที่สุดและกรณีที่แย่ที่สุด

	ProtosuffUnsafeAnalyzer (Best)	JavaSerializeUnsafeAnalyzer (Worst)
Clean - sentence	21 ms	1032 ms
Dirty - sentence	18 ms	5986 ms

จากตารางที่ 4.2 การวัดประสิทธิภาพการทำงานของโมดูลการตัดคำในกรณีที่ 2 นี้ เพื่อแสดงให้เห็นว่าการสร้างโปรแกรมการตัดคำภาษาไทยแบบคีย์เวิร์ดบนระบบการค้นหาข้อมูล หากไม่มีการทดสอบประสิทธิภาพการแปลงสายอักขระให้เป็นโครงสร้างข้อมูลเชิงวัตถุและโครงสร้างข้อมูลแบบทรัยที่จะนำมาใช้ในการสร้าง Analyzer กรณีที่ดีที่สุดและกรณีที่แย่ที่สุด ที่อาจเกิดขึ้นได้ในการตัดคำภาษาไทย จะมีเวลาในการทำงานที่แตกต่างกันอย่างชัดเจน ดังนั้น การทดสอบวัดประสิทธิภาพการทำงานของการแปลงโครงสร้างข้อมูลเชิงวัตถุให้เป็นสายอักขระและโครงสร้างข้อมูลแบบทรัยก่อนการสร้าง Analyzer เพื่อนำมาใช้ในการตัดคำภาษาไทย จะทำให้การตัดคำภาษาไทยมีความรวดเร็วมากยิ่งขึ้น

4.5 การวัดประสิทธิภาพการทำงานของการขยายและจำกัดเขต (Branch and Bound)

จากการนำเทคนิควิธีการขยายและจำกัดเขตมาใช้ในการเพิ่มประสิทธิภาพการตัดคำภาษาไทยของ safe segmentation ในโปรแกรมการตัดคำภาษาไทยแบบคีย์เวิร์ดบนระบบการค้นหาข้อมูล ซึ่งวิธีนี้จะสามารถหาคำตอบที่ดีที่สุดได้ในเวลาอันรวดเร็วกว่าหลักการทำงานการตัดคำในแบบการค้นหาตามแนวลึก (depth first search) และการค้นหาตามแนวกว้าง (breadth first search)

การวัดประสิทธิภาพการทำงานของการตัดคำที่ทำการเพิ่มประสิทธิภาพด้วยเทคนิควิธีการขยายและจำกัดเขต ผู้พัฒนาได้ทำการสร้าง Analyzer ขึ้นมา ประกอบด้วยการนำการแปลงสายอักขระให้เป็นโครงสร้างข้อมูลเชิงวัตถุที่มีเวลาในการทำงานเร็วที่สุด คือ Protostuff และโครงสร้างข้อมูลแบบทรัย คือ TrieA โดยใช้รูปแบบการตัดคำแบบ safe segmentation ของแต่ละเทคนิคที่นำมาใช้ในการตัดคำภาษาไทยมาเป็นตัววัดประสิทธิภาพการทำงาน ซึ่งมีทั้งหมด 3 Analyzer

- ProtostuffSafeDFSAnalyzer
  ประกอบด้วย
  Protostuff + TrieA + safe segment using depth first search
- ProtostuffSafeBFSAnalyzer
  ประกอบด้วย
  Protostuff + TrieA + safe segment using breadth first search
- ProtostuffSafeBnBAnalyzer
  ประกอบด้วย
  Protostuff + TrieA + safe segment using branch and bound

โดยนำ clean-sentence และ dirty-sentence เป็นข้อมูลที่จะนำมาใช้ในการวัดประสิทธิภาพการทำงานของ Analyzer แสดงผลข้อมูลได้ดังตาราง 4.3

ตารางที่ 4.3 แสดงประสิทธิเวลาการทำงานของ Analyzer รูปแบบ safe segment ที่ใช้เทคนิคการตัดคำแต่ละแบบ

	ProtostuffSafeDFSAnalyzer	ProtostuffSafeBFSAnalyzer	ProtostuffSafeBnBAnalyzer
Clean - sentence	2196 ms	4533 ms	22 ms
Dirty - sentence	44 ms	193 ms	9 ms

จากตารางที่ 4.3 การตัดคำภาษาไทยแบบคีย์เวิร์ดบนระบบการค้นหาข้อมูล รูปแบบ safe segmentation ที่ทำการเพิ่มประสิทธิภาพด้วยเทคนิควิธีการขยายและจำกัดเขต ทำให้เวลาการทำงานในการตัดคำภาษาไทยมีความรวดเร็วมากยิ่งขึ้นเมื่อเปรียบเทียบกับเทคนิคการตัดคำแบบการค้นหาตามแนวลึกและเทคนิคการตัดคำแบบการค้นหาตามแนวกว้าง อีกทั้งยังได้ผลลัพธ์ทุกความเป็นไปได้ของคำในแต่ละของประโยคด้วย

บทที่ 5
สรุปผลการดำเนินงาน และข้อเสนอแนะ

5.1 สรุปผลการดำเนินงาน

ในการปฏิบัติงานสหกิจครั้งนี้ได้รับมอบหมายให้ทำการพัฒนาการตัดคำภาษาไทยแบบคีย์เวิร์ดบนระบบการค้นหาข้อมูล (Thai word segment keyword on search engine) เป็นการตัดคำภาษาไทยเพื่อใช้เป็นดัชนีฐานข้อมูล ผลลัพธ์ถูกแบ่งออกเป็น 2 รูปแบบ คือ รูปแบบที่ 1 ประโยคที่ไม่มีคำสะกดผิดภายในประโยคและแสดงผลลัพธ์ทุกผลลัพธ์ที่เป็นไปได้ รูปแบบที่ 2 ประโยคที่มีคำสะกดผิดและแสดงผลลัพธ์เพียงผลลัพธ์เดียวที่มีคำขึ้นต้นของประโยคที่ยาวที่สุด ในกระบวนการทำงานได้ทำการพัฒนาฟังก์ชัน findAllPrefix ลงในโครงสร้างข้อมูลแบบทรัย ซึ่งเป็นฟังก์ชันหาคำขึ้นต้นทุกคำที่มีความหมายในประโยค เช่น คนแก่ขนของ คำขึ้นต้นที่มีความหมาย ได้แก่ คนแก่ และ คน เป็นต้น หลังจากนั้นนำคำขึ้นต้นแต่ละคำไปตัดประโยค เพื่อหาคำขึ้นต้นของประโยคย่อย ทำเช่นนี้ไปจนกว่าจะจบประโยคของคำขึ้นต้นนั้นของทุกคำขึ้นต้นผลลัพธ์จะเป็นทุกคำที่มีความหมายภายในประโยค

โครงสร้างข้อมูลแบบทรัย (Trie Structure) ถูกใช้ในการสร้างและเก็บคำศัพท์ในพจนานุกรม เพื่อนำมาใช้เป็นโครงสร้างข้อมูลหลักที่ใช้ในการสืบค้นคำ โดยมีการพัฒนาฟังก์ชันการหาคำขึ้นต้น findAllPrefix เพิ่มลงในโครงสร้างข้อมูลแบบทรัยเพื่อให้กระบวนการทำงานตรงตามขอบเขตความต้องการในการนำโครงสร้างข้อมูลแบบทรัยไปใช้งาน และมีการทดสอบ ดังนี้

1. วัดประสิทธิภาพเวลาการทำงานของโครงสร้างข้อมูลแบบทรัย โดยเลือกการทำงานของโครงสร้างข้อมูลแบบทรัยที่ใช้เวลารวดเร็วที่สุดและช้าที่สุดในการค้นหาข้อมูลร่วมกับฟังก์ชัน findAllPrefix

2. ไลบรารี่ Serialization ที่ทำงานร่วมกับโครงสร้างข้อมูลแบบทรัยที่มีคลังคำอยู่ภายในโครงสร้างทั้งหมด 40,000 คำ แล้วมีการทำงานเร็วที่สุดและช้าที่สุด

ซึ่งผลการทดสอบสรุปได้ว่าโครงสร้างข้อมูลแบบ TrieA ที่มีโครงสร้างแบบลิงค์ลิสต์และไลบรารี่ Protostuff ให้ผลลัพธ์ที่ดีที่สุด โครงสร้างข้อมูลแบบ TrieB ที่มีโครงสร้างแบบแฮชแม็บและไลบรารี่ Java serialize ให้ผลลัพธ์ที่แย่ที่สุด ดังนั้น โครงสร้างข้อมูลแบบ TrieA ร่วมกับ Protostuff และ โครงสร้างข้อมูลแบบ TrieB ร่วมกับ Java serialize มาใช้ในการสร้างโมดูลการตัดคำ ซึ่งในส่วนอัลกอริทึมการตัดคำนั้น โครงงานนี้นำเสนออัลกอริทึมในการตัดคำจำนวน 2 อัลกอริทึม คือ อัลกอริทึมแบบ Safe segmentation และอัลกอริทึมแบบ Unsafe segmentation ซึ่งอัลกอริทึม Safe segmentation ให้ผลลัพธ์เป็นคำทุกคำในประโยคที่มีอยู่ในพจนานุกรมซึ่งแตกต่างจากอัลกอริทึม Unsafe segmentation ที่ใช้ในการตัดคำของประโยคที่มีคำที่สะกดผิด หรือมีคำที่ไม่พบในพจนานุกรมและจะแสดงผลลัพธ์เพียง

หนึ่งผลลัพธ์เท่านั้น ทำให้ Safe segmentation มีเวลาการทำงานมากกว่า Unsafe segmentation หลายเท่า ดังนั้น จึงได้มีการเพิ่มประสิทธิภาพเวลาการทำงานของ Safe segmentation ให้รวดเร็วขึ้น โดยใช้เทคนิคแบบการขยายและจำกัดเขต (Branch and bound) ซึ่งผลการดำเนินงานสามารถทำงานได้ถูกต้องตามวัตถุประสงค์และขอบเขตที่กำหนดไว้

5.2 ข้อจำกัดของการพัฒนาและข้อเสนอแนะ

การตัดคำภาษาไทยแบบคีย์เวิร์ดบนระบบการค้นหาข้อมูล การวัดประสิทธิภาพการทำงานของโมดูลการตัดคำยังไม่สามารถวัดประสิทธิภาพในแง่ของความถูกต้องได้ เนื่องจากข้อมูลชุดทดสอบโปรแกรมการตัดคำภาษาไทย BEST2009 ของศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ (NECTEC) ไม่ตรงกับขอบเขตการตัดคำภาษาไทยของโครงงานนี้ ทำให้ผลการทดสอบที่ได้จากการตัดคำภาษาไทยเทียบกับข้อมูลชุดทดสอบยังไม่สามารถตรวจสอบในแง่ของความถูกต้องได้และการตัดคำภาษาไทยแบบคีย์เวิร์ดบนระบบการค้นหาข้อมูล ยังมีข้อจำกัดเรื่องของคำศัพท์ในพจนานุกรม เนื่องจากการตัดคำภาษาไทยแบบคีย์เวิร์ดบนระบบการค้นหาข้อมูลเป็นการตัดคำภาษาไทยแบบพจนานุกรม ถ้าหากประโยคหรือบทความที่รับเข้ามาทำการตัดคำภาษาไทยมีคำบางคำที่ไม่พบอยู่ในพจนานุกรม จะส่งผลให้ผลลัพธ์ของการตัดคำภาษาไทยที่ได้มาไม่ถูกต้อง

อนาคตจึงต้องมีการพัฒนาและปรับปรุงพจนานุกรมให้สามารถเพิ่มขอบเขตของคำศัพท์ให้มีมากยิ่งขึ้นเพื่อครอบคลุมคำศัพท์ที่นำมาใช้ในการตัดคำภาษาไทยและพัฒนาโปรแกรมการตัดคำภาษาไทยแบบคีย์เวิร์ดบนระบบการค้นหาข้อมูลให้สามารถแบ่งตัวเลข อักขระพิเศษและคำศัพท์ภาษาอังกฤษได้อย่างถูกต้องตามข้อมูลที่ทำการรับเข้ามา

เอกสารอ้างอิง

[1] **Introduction to Information Retrieval.** 2009. [Online]. Available : http://nlp.stanford.edu/IR-book/pdf/02voc.pdf. เข้าถึงเมื่อวันที่

[2] Surapant Meknavin, Paisarn Charoenpornsawat and Boomserm Kijsirikul. 1998. **Feature-based Thai Word Segmentation.** [Online]. Available : http://www.cs.cmu.edu/~paisarn/papers/nlprs97.pdf. เข้าถึงเมื่อวันที่

[3] นายไพศาล เจริญพรสวัสดิ์. 2541. **การตัดคำภาษาไทยโดยใช้คุณลักษณะ.** [Online]. Available : http://www.cs.cmu.edu/~paisarn/papers/thesis99.pdf. เข้าถึงเมื่อวันที่

[4] ไกรศักดิ์ เกสร. 2554. **การค้นหาข้อมูลเชิงความหมาย:แนวคิดใหม่ของโปรแกรมการค้นหา (Search Engine) และแนวทางการพัฒนาในอนาคต.** [Online]. Available : http://acad.vru.ac.th/Journal/01_1-2.pdf. เข้าถึงเมื่อวันที่ 9 ม.ค. 2560.

[5] สมชาย ประสิทธิ์จูตระกูล. 2553. **การออกแบบและวิเคราะห์อัลกอริทึม.** พิมพ์ครั้งที่ 4. กรุงเทพฯ : โรงพิมพ์จุฬาลงกรณ์มหาวิทยาลัย.

[6] โอภาส เอี่ยมสิริวงศ์. 2549. **โครงสร้างข้อมูล (Data Structures) เพื่อการออกแบบโปรแกรมคอมพิวเตอร์.** กรุงเทพฯ : ซีเอ็ดยูเคชั่น.

[7] วีระศักดิ์ ซึงถาวร. 2548. **Java programming volume II.** กรุงเทพฯ : ซีเอ็ดยูเคชั่น.

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

[9] เนรมิต ชุมสาย ณ อยุธยา. 2559. **Recursion.** [Online]. Available : http://sci.feu.ac.th/faa/dsa/bookPDFs/chap4-Recursion.pdf. เข้าถึงเมื่อวันที่ 30 ส.ค. 2559.

[10] Supasiri.s. 2559. **WHAT IS SOFTWARE PERFORMANCE TESTING?.** [Online]. Available : http://www.aware.co.th/software-performance-testing/. เข้าถึงเมื่อวันที่ 1 ก.ย. 2559

[11] **ความรู้พื้นฐานเกี่ยวกับโครงสร้างข้อมูลและอัลกอริธึม.** 2559. [Online]. Available : http://www.mwit.ac.th/~jeab/sheet40206/IT40206-Intro.pdf. เข้าถึงเมื่อวันที่ 1 ก.ย. 2559

เอกสารอ้างอิง (ต่อ)

[12] **Recurrence relation.** 2559. [Online]. Available : https://en.wikipedia.org/wiki/Recurrence_relation#See_also. เข้าถึงเมื่อวันที่ 30 ส.ค. 2559.

[13] **โปรแกรม Eclipse.** 2014. [Online]. Available : http:// www.itgenius.co.th/article/%E0%B9%82%E0%B8%9B%E0%B8%A3%E0%B9%81%E0%B8%81%E0%B8%A3%E0%B8%A1%20Eclipse.html. เข้าถึงเมื่อวันที่

[14] **EGit.** n.d. [Online]. Available : http://www.eclipse.org/egit/. เข้าถึงเมื่อวันที่

[15] **Charoenpornsawat, P.** (1999). Feature-based Thai Word Segmentation. Master's Thesis. Computer Engineering. Chulalongkorn University, Bangkok,Thailand.

ภาคผนวก

ภาคผนวก ก

การติดตั้ง Java JDK

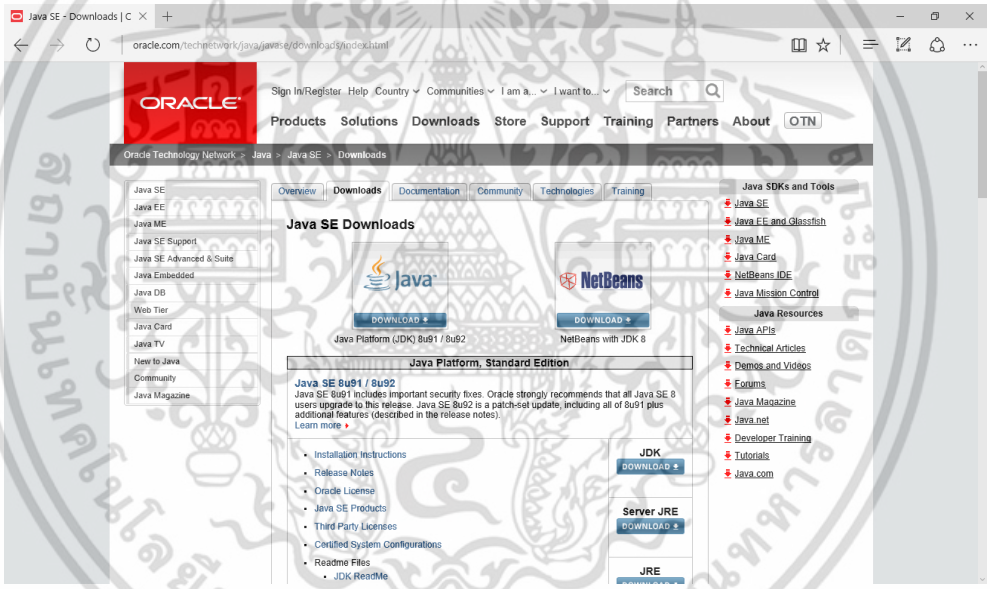
การติดตั้ง Java JDK

Java JDK เนื่องจากความต้องการของระบบที่ใช้ในโปรแกรม Eclipse ต้องการ Java JDK ในการติดตั้งโปรแกรมดังนั้นจึงควรติดตั้ง Java JDK ให้เรียบร้อยก่อน ซึ่งมีขั้นตอนการติดตั้งดังนี้

1) ดาวโหลดตัวติดตั้ง JRE มาไว้ที่เครื่อง โดยสามารถเลือกได้ว่าจะติดตั้งแต่ JRE หรือจะติดตั้งเป็น JDK (Java development Kit) โดยภายใน JDK จะมี JRE อยู่ด้วย แนะนำให้ติดตั้งเป็น JDK จะดีกว่าเพราะจะทำให้ Eclipse สามารถเขียน Application ที่เป็นภาษา Java ได้อีกด้วย

ดาวน์โหลดตาม Link ด้านล่างเลือก version ตามต้องการ

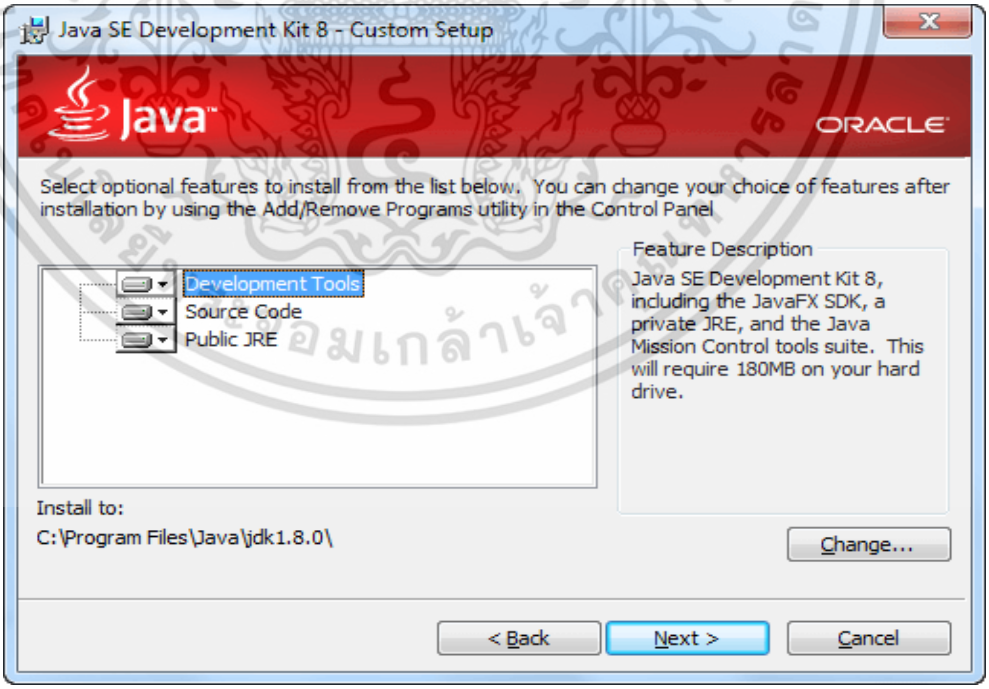
http://www.oracle.com/technetwork/java/javase/downloads/index.html



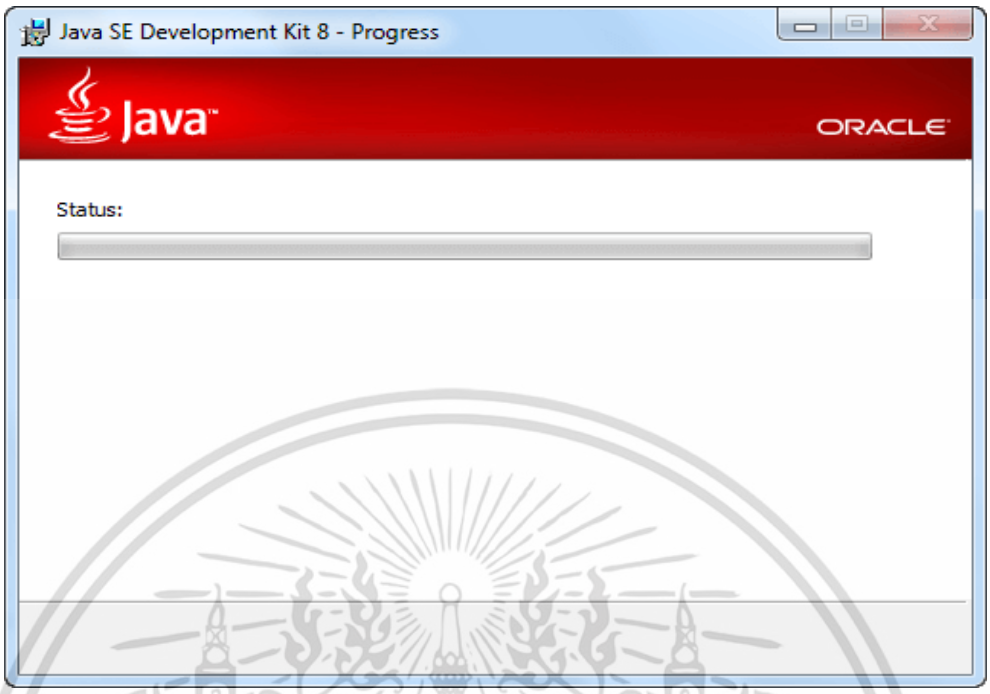
รูปที่ ก.1 หน้าเว็บไซต์ Download Java JDK

2) ทำการติดตั้งตามขั้นตอนดังรูปที่ ก.2 – ก.7

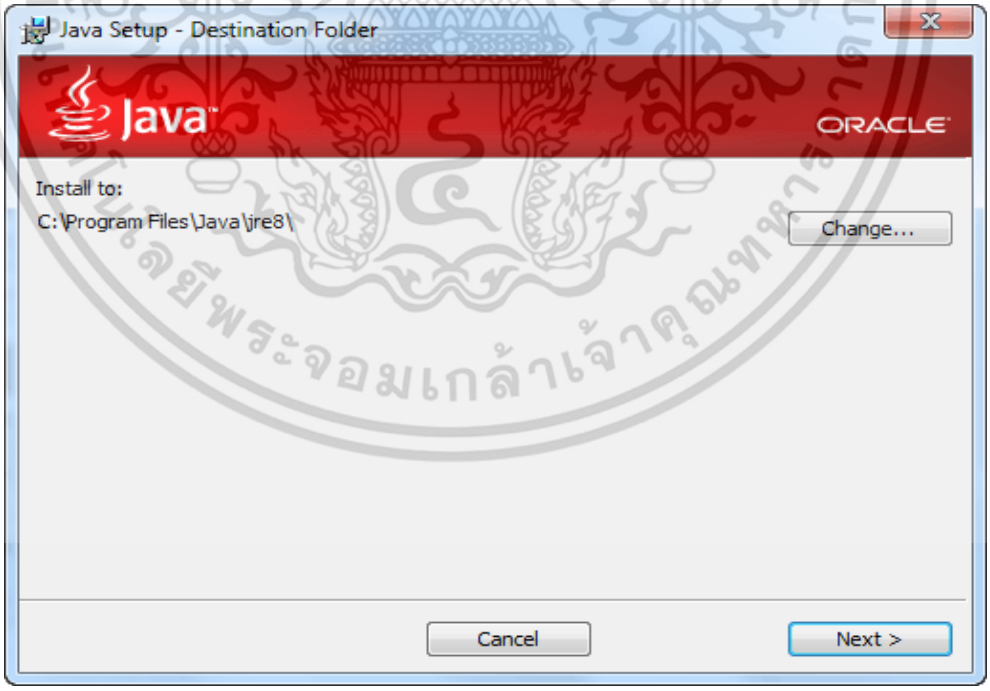
รูปที่ ก.2 ขั้นตอนการติดตั้ง Java JDK (1)



รูปที่ ก.3 ขั้นตอนการติดตั้ง Java JDK (2)

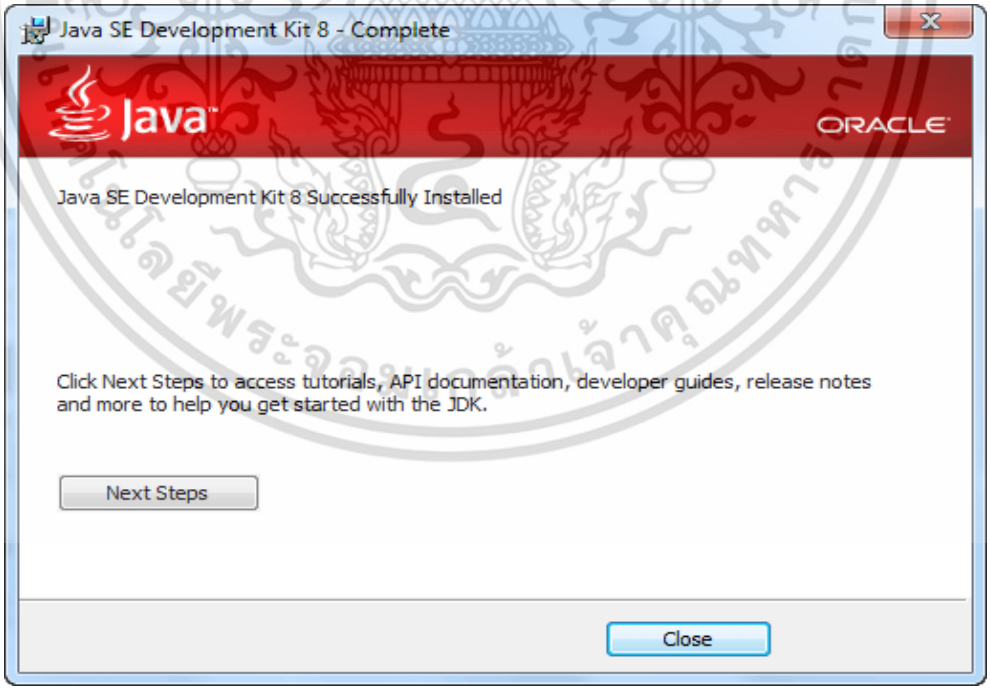


รูปที่ ก.4 ขั้นตอนการติดตั้ง Java JDK (3)



รูปที่ ก.5 ขั้นตอนการติดตั้ง Java JDK (4)

รูปที่ ก.6 ขั้นตอนการติดตั้ง Java JDK (5)



รูปที่ ก.7 ขั้นตอนการติดตั้ง Java JDK (6)

ภาคผนวก ข

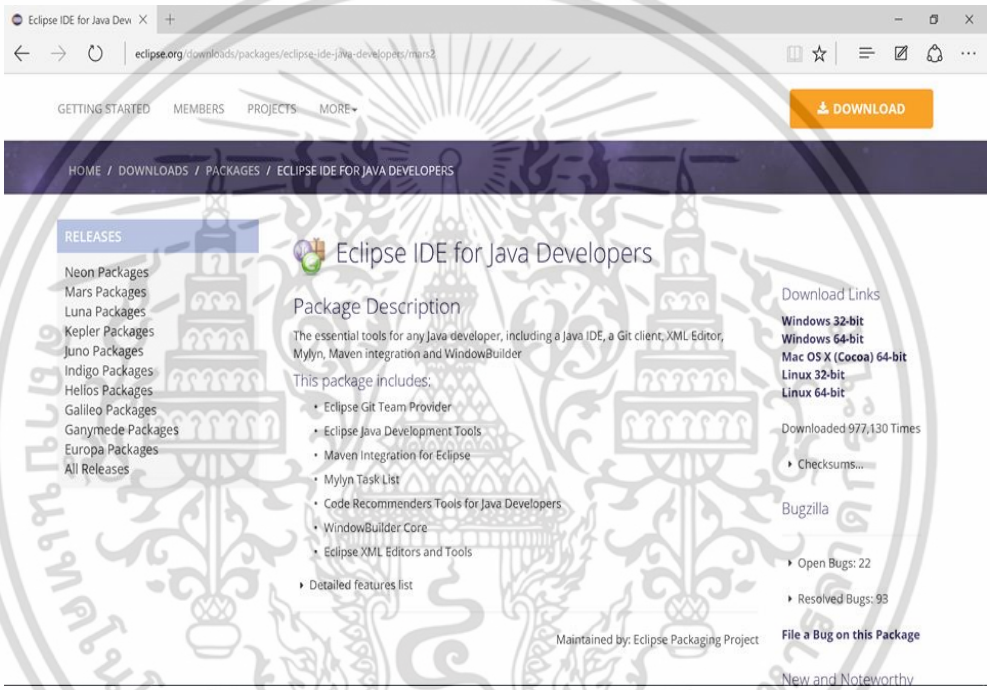
การติดตั้ง Eclipse

การติดตั้ง Eclipse

Eclipse เป็นโปรแกรมสำหรับพัฒนาที่รองรับหลายภาษา แต่สำหรับโครงการนี้เราจะนำมาใช้พัฒนาโครงการด้วยภาษาจาวา ซึ่งมีวิธีการติดตั้ง Eclipse บน Windows ดังขั้นตอนการติดตั้งต่อไปนี้

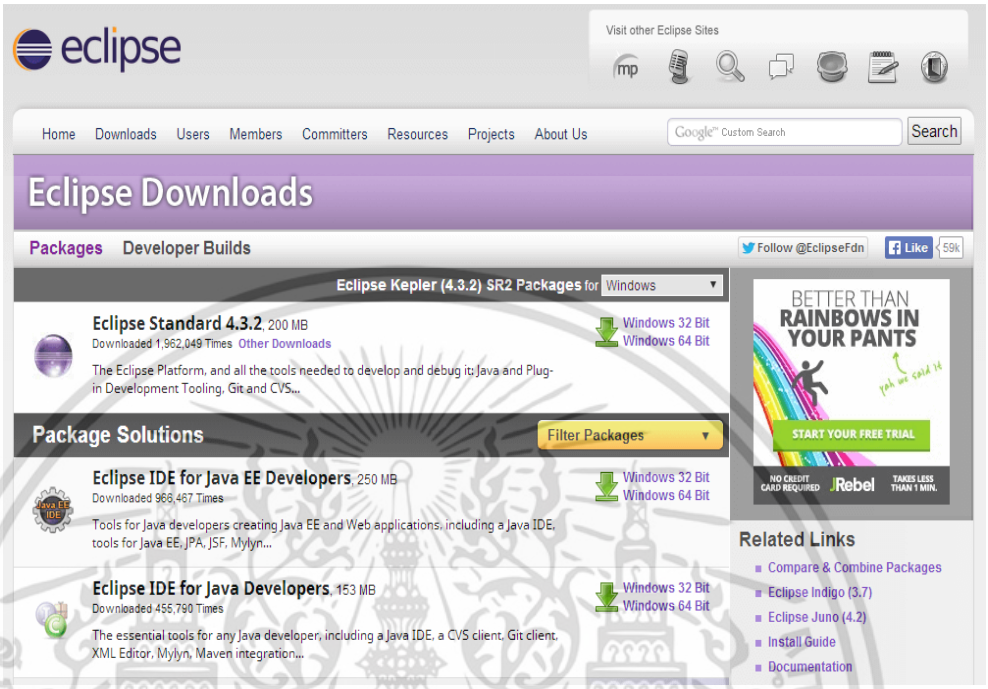
1) ดาวน์โหลดตัวติดตั้ง **Eclipse**

https://www.eclipse.org/downloads/



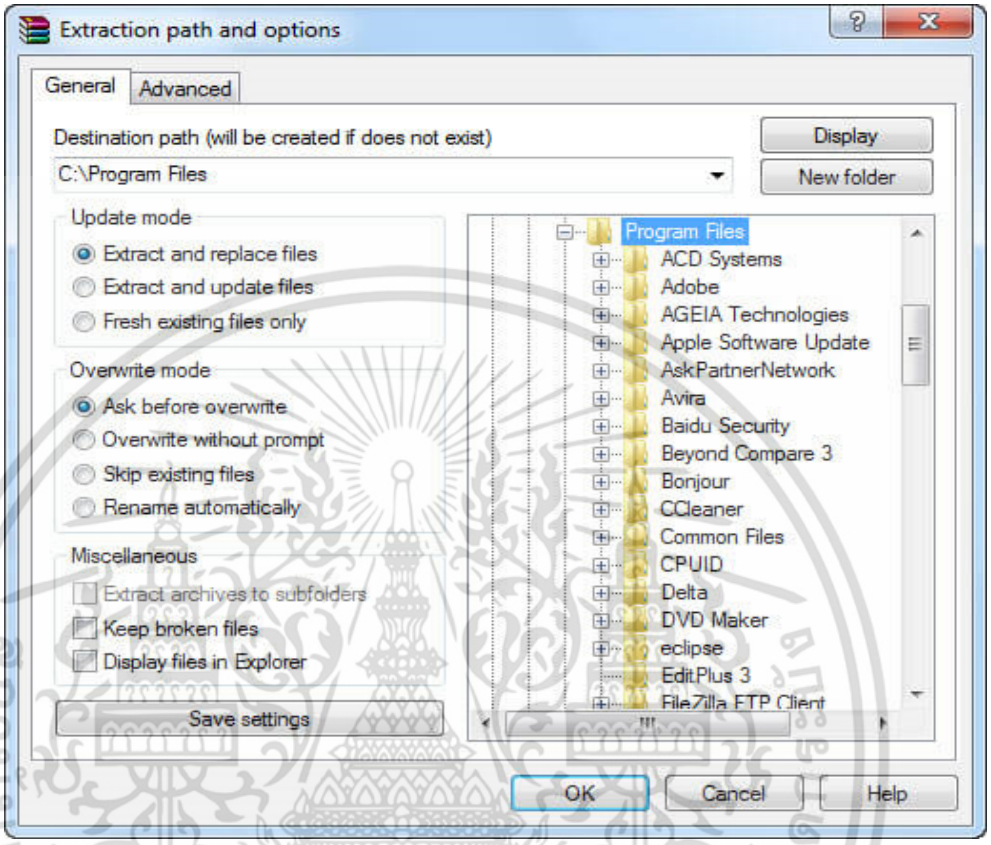
รูปที่ ข.1 หน้าเว็บไซต์ Download Eclipse

2) เลือกดาวโหลด version ตามต้องการ

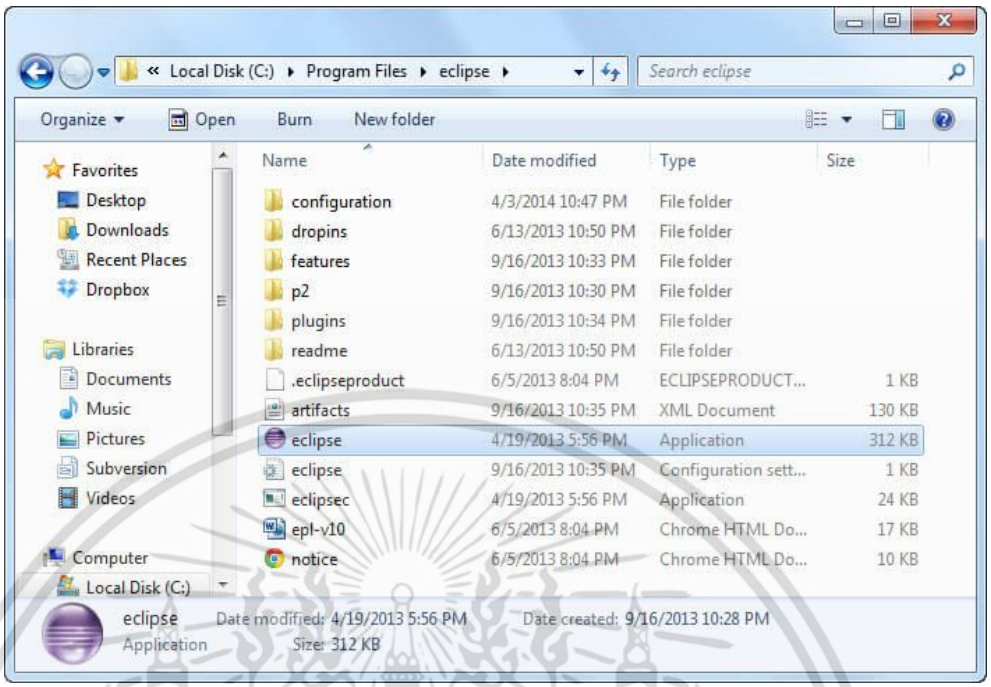


รูปที่ ข.2 หน้าเว็บไซต์ Download Eclipse เลือก version

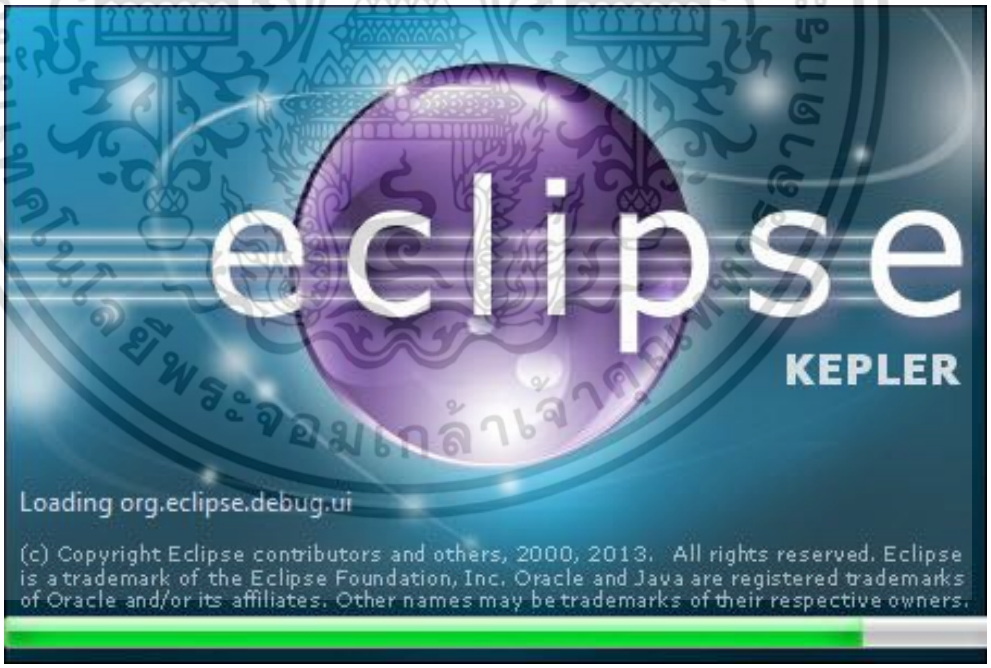
3) ทำการติดตั้งตามขั้นตอนดังรูปที่ ข.3 - ข.6



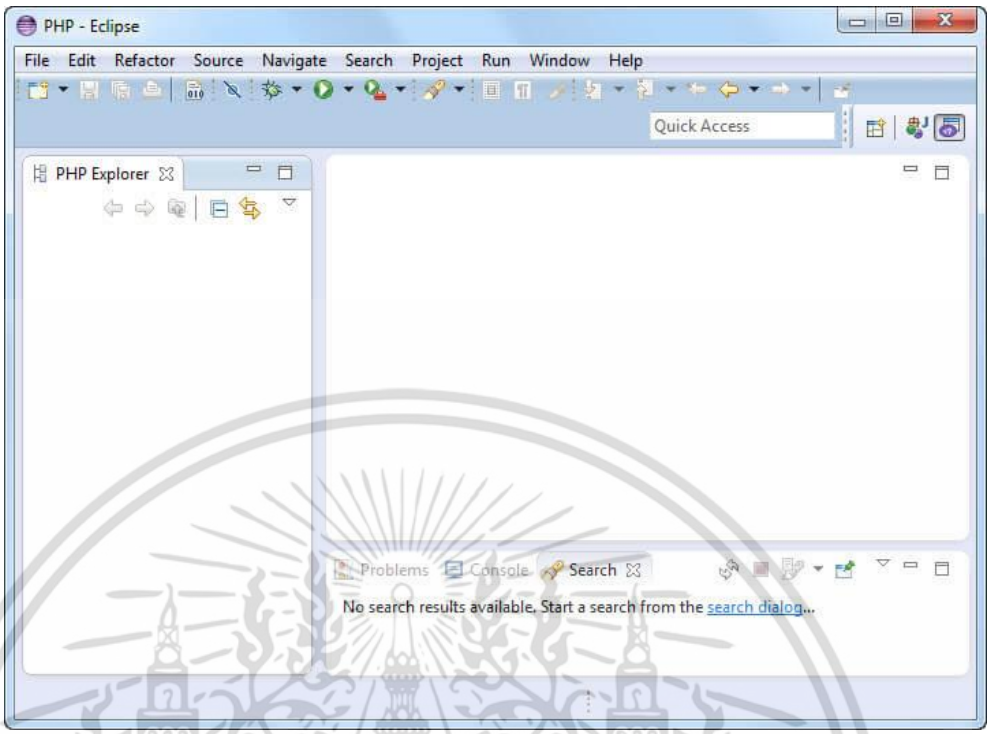
รูปที่ ข.3 ขั้นตอนการติดตั้ง Eclipse (1)



รูปที่ ข.4 ขั้นตอนการติดตั้ง Eclipse (2)



รูปที่ ข.5 ขั้นตอนการติดตั้ง Eclipse (3)



รูปที่ ข.6 ขั้นตอนการติดตั้ง Eclipse (4)

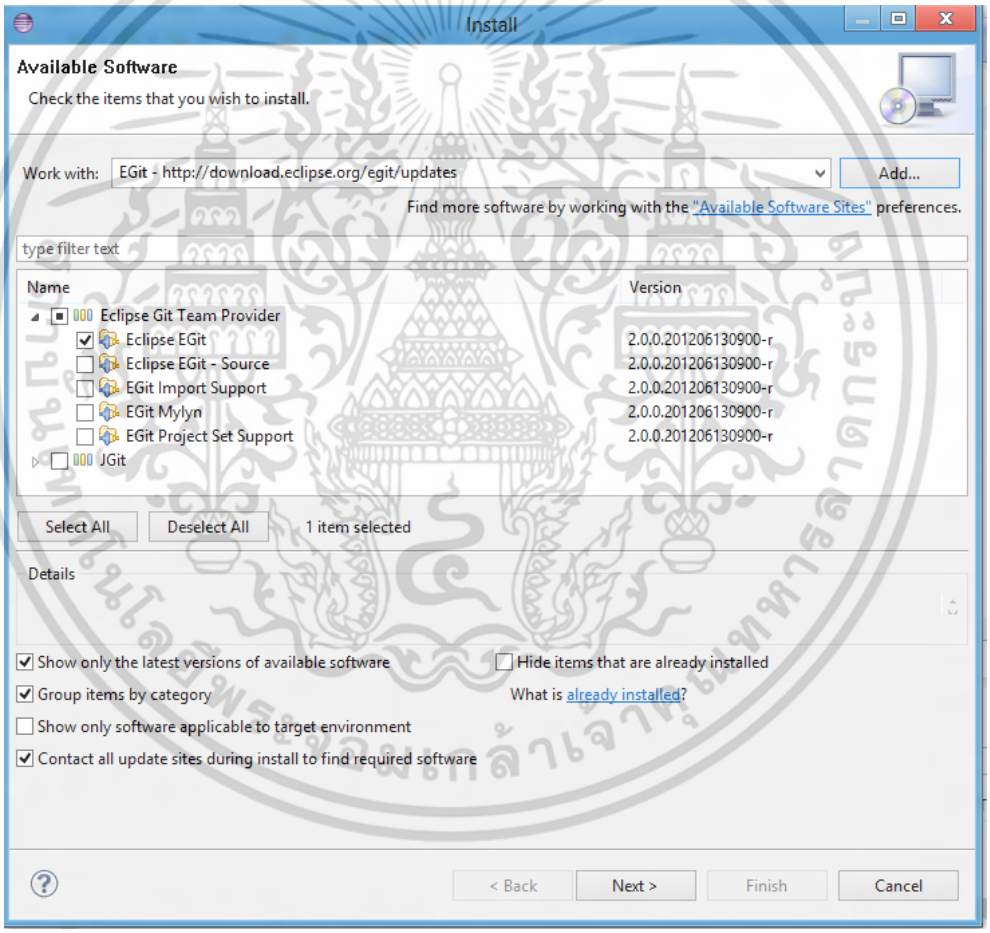
ภาคผนวก ค

การติดตั้ง EGit

การติดตั้ง EGit

การพัฒนาโครงการบนโปรแกรม Eclipse มีการใช้นำ Git เข้ามาใช้งาน เพื่อป้องกันการสูญหาย แก้ไปแล้วลืมว่าลบอะไรทิ้งไป แก้ผิดแล้วโค้ดพังหมด หรือเหตุสุดวิสัยอย่างอื่นขณะพัฒนาโครงการ ดังนั้นเราจึงได้นำ Git มาช่วยในการพัฒนา ซึ่งมีขั้นตอนการติดตั้งดังนี้

- ลง Plugin สำหรับ Git ก่อน นามว่า "EGit" เลือก Windows -> Install new software ใส่ URL ในลงในช่อง http://download.eclipse.org/egit/updates เลือกแค่ Eclipse EGit ก็เพียงพอแล้ว แล้วก็รอจนมันลงเสร็จ

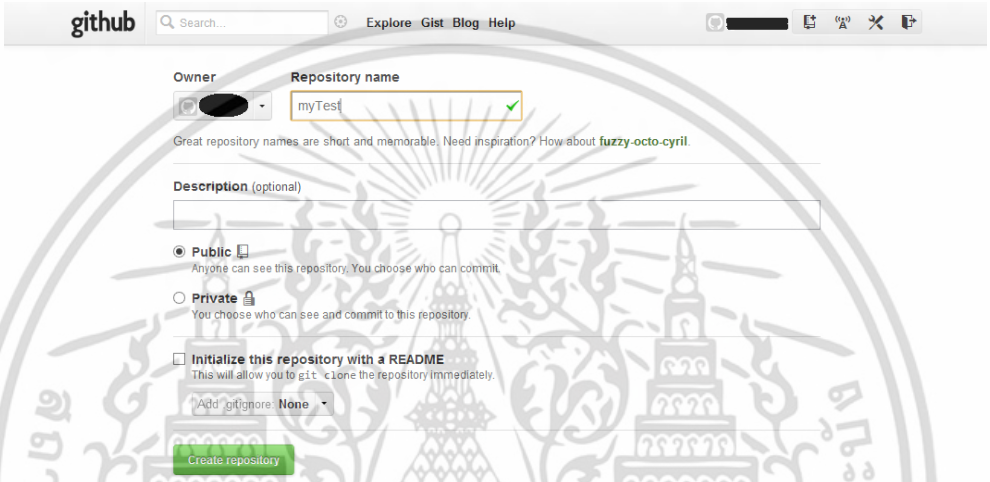


รูปที่ ค.1 การติดตั้ง Plugin Git บน Eclipse

ในส่วนของการใช้ จะแบ่งเป็น 2 กรณี

กรณีแรก อยากมี repository เป็นของตัวเอง (อยากเป็นผู้ริเริ่ม)

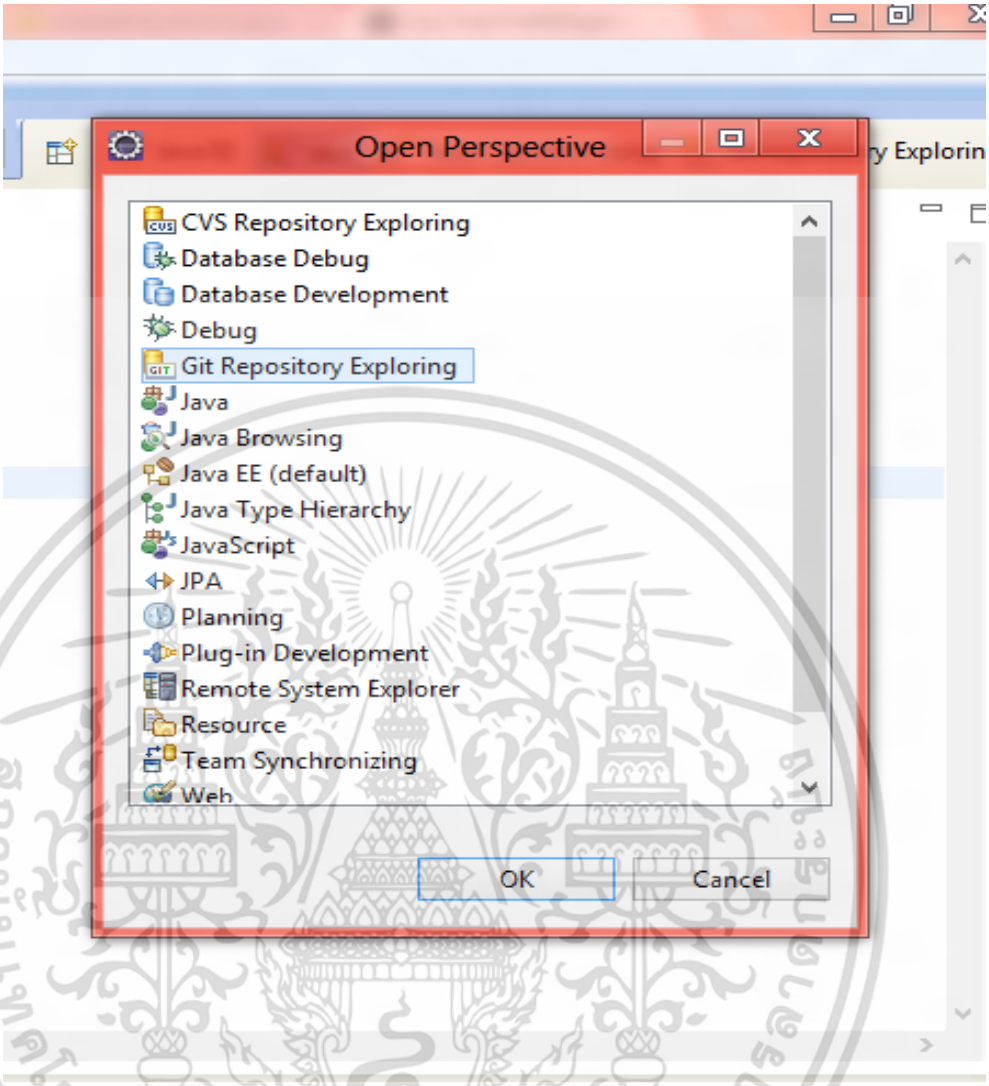
1) สร้าง repository บน GitHub ก่อน กดปุ่ม new repository จากนั้นตั้งชื่อ Repository name ไป แล้วก็กด create ได้เลย



รูปที่ ค.2 สร้าง repository บน GitHub

เสร็จแล้วจะได้ Repository ว่างๆ มา หน้าเว็บจะมีหน้าตาเป็นตัวหนังสือ ซึ่งเป็น code ของ git

2) เมื่อเราสร้าง Repository แล้ว เราต้องทำให้ Eclipse รู้จักมัน โดย Open Perspective ( ) แล้วเลือก Git Repository

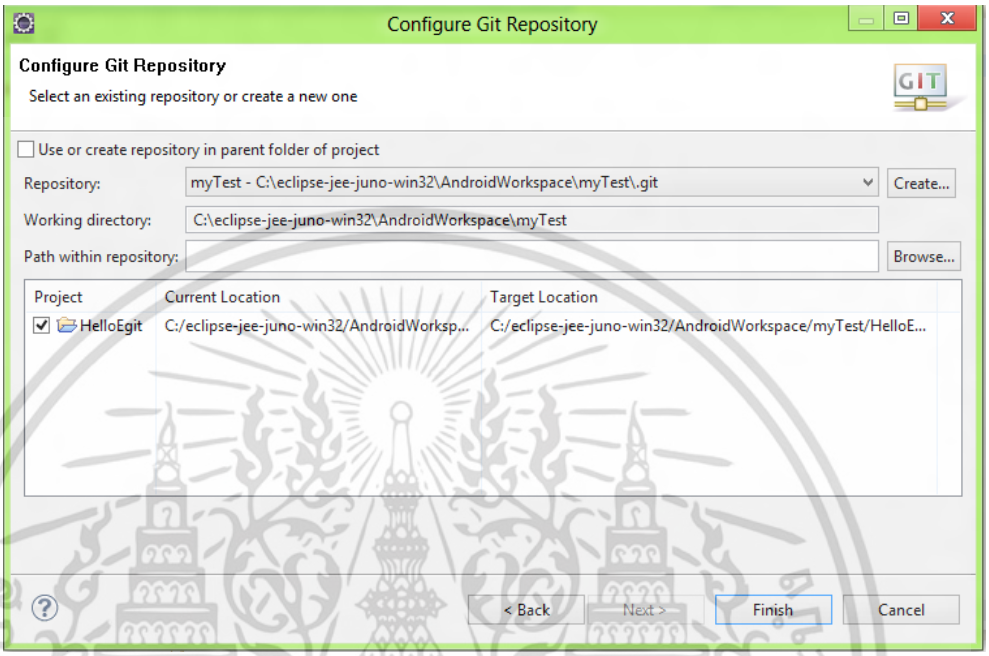


รูปที่ ค.3 Open Perspective บน GitHub

จากนั้นกด Clone โดยไป copy link ที่เป็น .git มาจากบนเว็บ

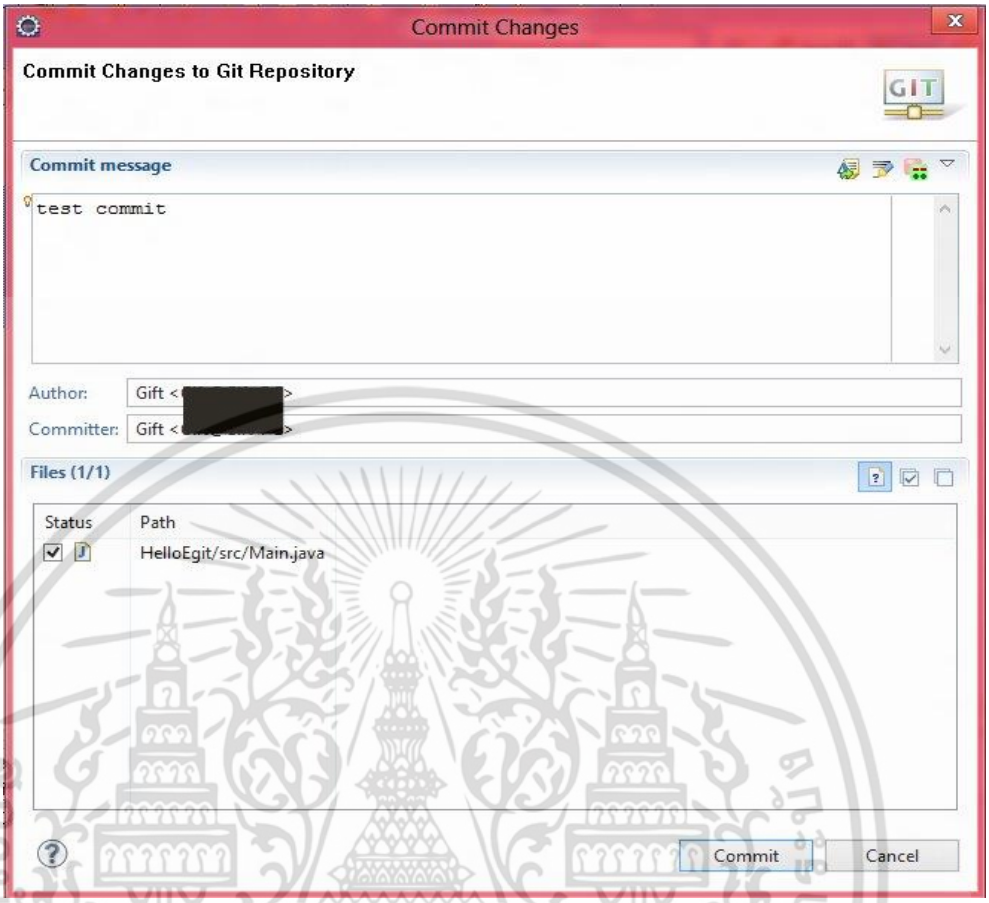
รูปที่ ค.4 ทำการ Clone git

3) เนื่องจากเป็นการ Repository ตัวใหม่ เราก็ต้องมี code อยู่ในเครื่องเราก่อนแล้ว หลังจากนั้น คลิกขวาที่โปรเจค เลือก Team > Share Project แล้วเลือกไปที่ Repository ที่สร้างไว้ในข้อ 2)



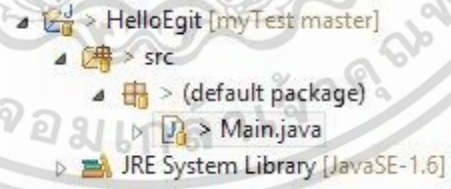
รูปที่ ค.5 นำ source code โปรเจคเข้า Git

4) จากนั้นก็จะสามารถ commit ได้ โดยไปที่ Team -> Commit *ต้องใส่ comment ด้วย



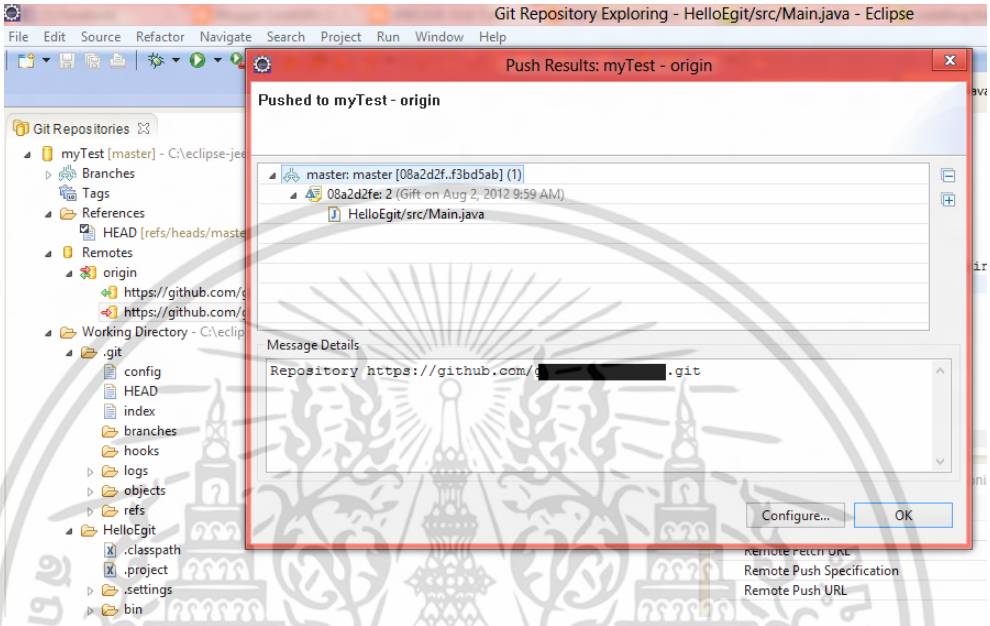
รูปที่ ค.6 การ commit code

*ถ้าไฟล์ยังไม่ commit จะมีรูป > ตามภาพข้างล่าง



รูปที่ ค.7 commit ไม่สำเร็จ

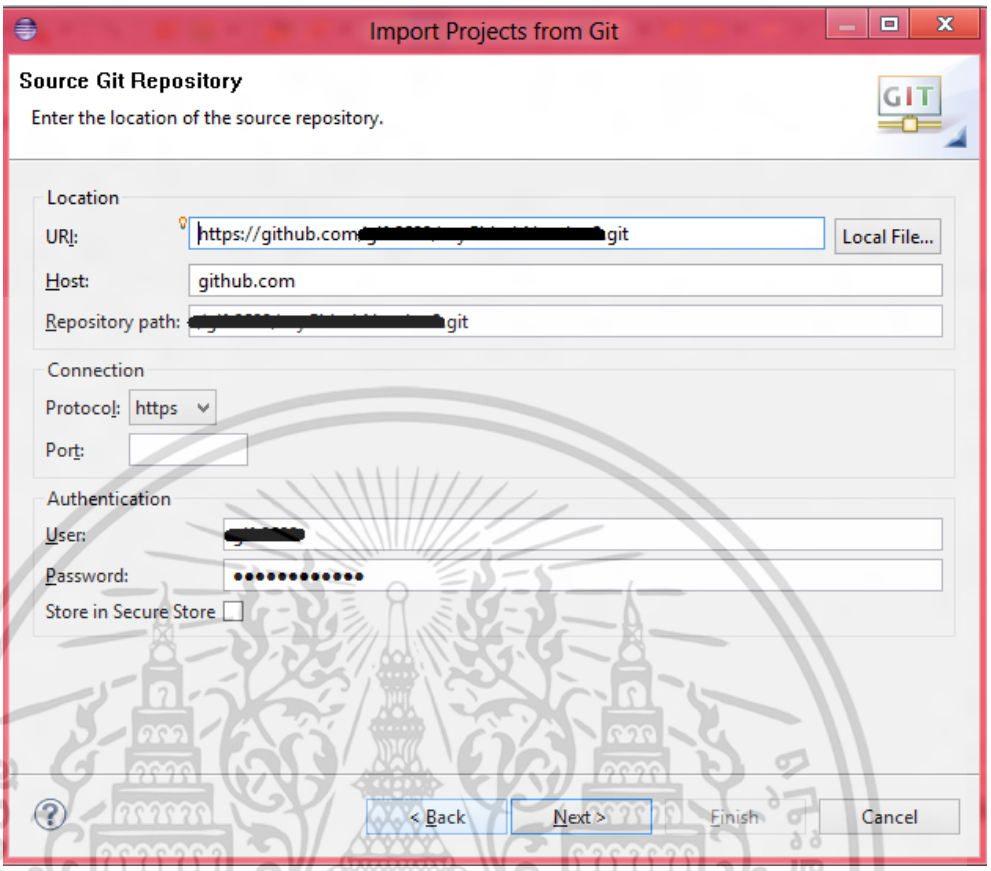
5) ถ้าทำงานเสร็จแล้ว ก็ทำการ push เพื่อเป็นการส่งทุกอย่างไปที่ respository ของเรา (ไปที่ perspective git -> repo. ของเรา -> remote -> origin -> คลิกขวา push)



รูปที่ ค.8 การ push ลง Git

กรณีที่สอง การ Clone ของคนอื่นมาใช้งาน

1) File -> Import -> Import Project from Git -> URI จากนั้นเอา link repository มาแปะ ใส่ username , password ด้วย เพราะว่าบาง Git จะไม่อนุญาตให้ดึงได้เลย

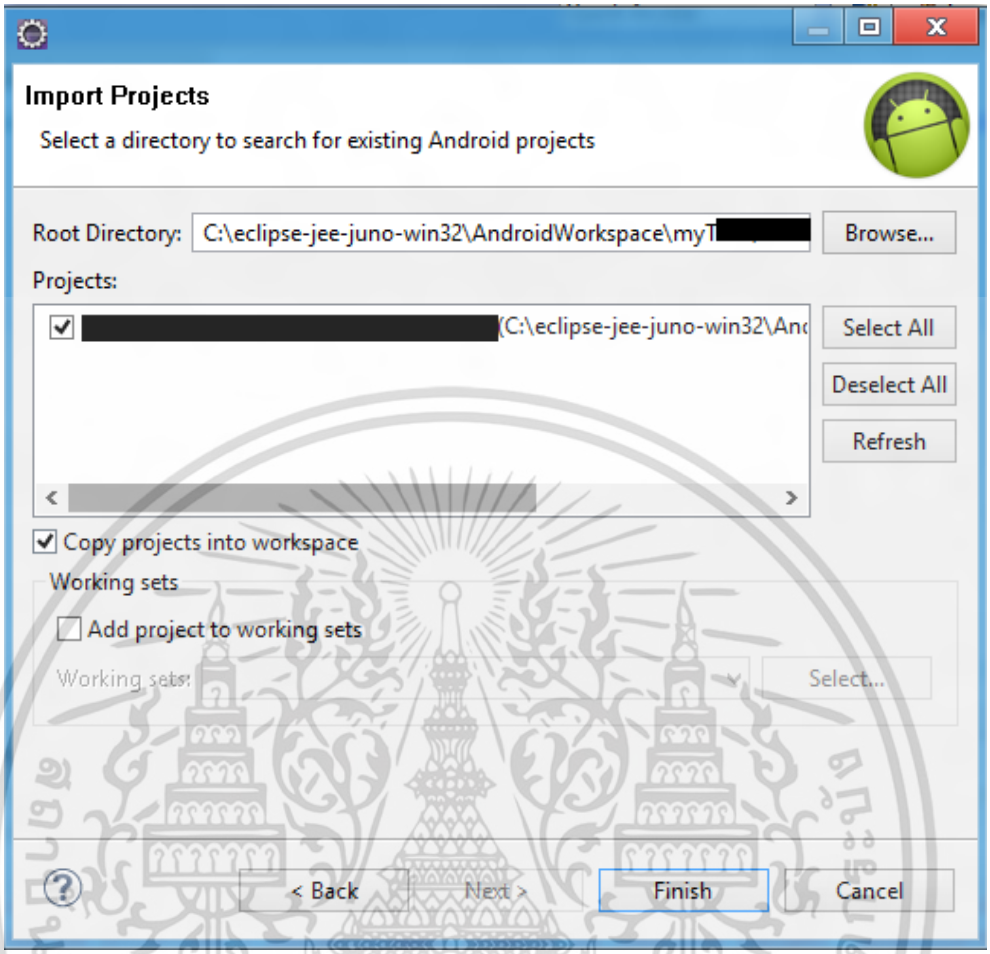


รูปที่ ค.9 การ Clone Git

จากตรงนี้ Eclipse ก็จะรู้จักกับ Repository แล้ว หมายความว่า source code อยู่ที่เครื่องเราเป็นที่เรียบร้อย

2) ในส่วนนี้ ก็สามารถ import project เข้ามาได้เลย ในกรณีตัวอย่าง New -> Android Project From Existing Code เพราะโปรเจคเป็น Android **ไม่ว่าโปรเจคของเราเป็นอะไรก็แล้วแต่ จะเจอหน้าตา Import Projects นี้ครับ ให้ Browse ไปที่ path Repository ของเรา และหาโปรเจคที่ต้องการ Import

*การติ๊ก Copy projects into workspace เพื่อให้ Eclipse copy โค้ดของเราออกมา



รูปที่ ค.10 import project

3) ทีนี้เราจะได้โปรเจคมาแล้ว แต่มันอาจจะยังไม่สามารถ sync. ได้ ก็ต้อง Share Project ตามวิธีด้านบน (ข้อ 3) สำหรับคนที่ติ๊ก Copy projects into workspace ให้เลือก path within repository ไปที่เดียวกับข้อ 2 ด้วย