การพัฒนาเทคนิคสร้างดรรชนีสำหรับข้อมูลขนาดใหญ่ที่อยู่ในรูปแบบวัตถุของจาวาสคริปต์

จิรวัฒน์ ดวงแก้ว

วิทยานิพนธ์เสนอมหาวิทยาลัยพะเยา เพื่อเป็นส่วนหนึ่งของการศึกษา
หลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์
กุมภาพันธ์ 2567

ลิขสิทธิ์เป็นของมหาวิทยาลัยพะเยา

การพัฒนาเทคนิคสร้างดรรชนีสำหรับข้อมูลขนาดใหญ่ที่อยู่ในรูปแบบวัตถุของจาวาสคริปต์

จิรวัฒน์ ดวงแก้ว

วิทยานิพนธ์เสนอมหาวิทยาลัยพะเยา เพื่อเป็นส่วนหนึ่งของการศึกษา
หลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์
กุมภาพันธ์ 2567
ลิขสิทธิ์เป็นของมหาวิทยาลัยพะเยา

THE DEVELOPMENT OF INDEXING TECHNIQUES FOR LARGE SCALE DATA IN THE JAVASCRIPT OBJECT FORMAT

JIRAWAT DUANGKAEW

A Thesis Submitted to University of Phayao

in Partial Fulfillment of the Requirements

for the Master of Engineering Degree in Computer Engineering

February 2024

Copyright 2024 by University of Phayao

วิทยานิพนธ์

เรื่อง

การพัฒนาเทคนิคสร้างดรรชนีสำหรับข้อมูลขนาดใหญ่ที่อยู่ในรูปแบบวัตถุของจาวาสคริปต์

ของ จิรวัฒน์ ดวงแก้ว

ได้รับพิจารณาอนุมัติให้เป็นส่วนหนึ่งของการศึกษา

หลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์

ของมหาวิทยาลัยพะเยา

..... ประธานกรรมการสอบวิทยานิพนธ์

(ผู้ช่วยศาสตราจารย์ ดร. วีรพันธุ์ ศิริฤทธิ์)

..... ประธานที่ปรึกษาวิทยานิพนธ์

(ผู้ช่วยศาสตราจารย์ ดร. บวรศักดิ์ ศรีสังสิทธิสันติ)

..... อาจารย์บัณฑิตศึกษามหาวิทยาลัยพะเยา

(ผู้ช่วยศาสตราจารย์ ดร. พรเทพ โรจนวสุ)

..... อาจารย์บัณฑิตศึกษามหาวิทยาลัยพะเยา

(รองศาสตราจารย์ ดร. สาคร เมฆรักษาวนิช)

..... คณบดีคณะเทคโนโลยีสารสนเทศและการสื่อสาร

(ผู้ช่วยศาสตราจารย์ ดร. พรเทพ โรจนวสุ)

เรื่อง: การพัฒนาเทคนิคสร้างดรรชนีสำหรับข้อมูลขนาดใหญ่ที่อยู่ในรูปแบบวัตถุของจาวาสคริปต์

ผู้วิจัย: จิรวัฒน์ ดวงแก้ว, วิทยานิพนธ์: วศ.ม.(วิศวกรรมคอมพิวเตอร์), มหาวิทยาลัยพะเยา, 2566

อาจารย์ที่ปรึกษา: ผู้ช่วยศาสตราจารย์ ดร. บวรศักดิ์ ศรีสังสิทธิสันติ

คำสำคัญ: การจัดทำดรรชนีหนาแน่น, การทำดรรชนีกระจาย, ชุดข้อมูลขนาดใหญ่, ฐานข้อมูลที่ไม่ใช่เชิงสัมพันธ์, ไฟล์รูปแบบวัตถุของจาวาสคริปต์

บทคัดย่อ

การใช้รูปแบบวัตถุของจาวาสคริปต์ (JavaScript Object Notation, JSON) ในฐานข้อมูลที่ไม่ใช่เชิงสัมพันธ์ (Not only Structured Query Language, NoSQL) ได้รับความนิยมเป็นอย่างมาก อย่างไรก็ตาม หากพิจารณาข้อจำกัด NoSQL โดยเฉพาะส่วนของเทคนิคการจัดทำดรรชนีแบบหนาแน่น (Dense Index) แบบหนึ่งต่อหนึ่ง (One-to-One) และดรรชนีกระจาย (Sparse Index) แบบหนึ่งต่อกลุ่ม (One-to-Many) สำหรับชุดข้อมูลไฟล์ JSON ขนาดใหญ่ จึงเป็นข้อจำกัดอันท้าทายเป็นอย่างมาก งานวิจัยครั้งนี้ได้เสนอการพัฒนาเทคนิคสร้างดรรชนีสำหรับข้อมูลขนาดใหญ่ที่อยู่ในรูปแบบวัตถุของจาวาสคริปต์ (The development of Indexing techniques for large-scale data in the JavaScript object format) ผู้วิจัยได้จำลองข้อมูลไม่ต่ำกว่าแปดแสนรายการขึ้นไป ซึ่งมีขนาดไฟล์ข้อมูลไม่ต่ำกว่าสามสิบกิ๊กกะไบต์ อยู่ภายใต้ชนิดไฟล์ JSON นำเข้าสู่กระบวนการวิเคราะห์ออกแบบขั้นตอนวิธี โดยผสมผสานบางเทคนิคของฐานข้อมูลเชิงสัมพันธ์ (Relational databases) ได้แก่ ดรรชนีแบบหนาแน่น (Dense Index) และดรรชนีแบบกระจาย (Sparse Index) และการประยุกต์ใช้โครงสร้างแบบแถวลำดับ (Array) เทคนิคการค้นหาเชิงเส้น (Linear search, LS) เทคนิคการค้นหาแบบทวิภาค (Binary search, BS) เทคนิคต้นไม้ค้นหาแบบทวิภาค (Binary search Tree, BST) และเทคนิค Adelson-Velskii Landis Tree (AVL Tree) รวมถึงนำภาษาไพทอน (Python) สำหรับทดสอบเปรียบเทียบประสิทธิภาพทางด้านระยะเวลาของดรรชนีในการสืบค้นหาข้อมูล ผู้วิจัยได้นำเทคนิคข้างต้นเข้าสู่กระบวนการพัฒนาเทคนิคสร้างดรรชนีสำหรับข้อมูลขนาดใหญ่ที่อยู่ในรูปแบบวัตถุของจาวาสคริปต์ ได้แก่ ขั้นตอนวิธีสำหรับการจัดทำดรรชนีแบบหนาแน่นและดรรชนีแบบกระจาย ขั้นตอนเทคนิคการค้นหาเชิงเส้น (Linear search, LS) จากนั้นดำเนินการปรับแต่งโครงสร้างต้นไม้ให้เหมาะสมกับชุดข้อมูลของเทคนิค BS BST และ AVL Tree ซึ่งอยู่ภายใต้ชุดข้อมูลไฟล์ JSON ขนาดใหญ่ ท้ายที่สุดผลการวิจัยพบว่าแนวทางการจัดทำดรรชนีแบบหนาแน่น (Dense Index) สามารถลดเวลาในการเข้าถึงข้อมูลถึง 98.57% จากวิธีการไม่มีดรรชนี (Non Index) และดรรชนีแบบกระจาย (Sparse Index) สามารถลดเวลาในการเข้าถึงข้อมูลถึง 98.45% นอกจากนี้เมื่อเพิ่มประสิทธิภาพการเข้าถึงข้อมูลต่อคำหลักด้วยเทคนิค LS BS BST และ AVL Tree ผลการวิจัยพบว่าเทคนิค AVL Tree มีประสิทธิภาพเวลาที่เร็วที่สุดในการค้นหาต่อคำหลัก สำหรับกรณีดรรชนีแบบหนาแน่น โดยใช้เวลาเฉลี่ยเท่ากับ 0.005 มิลลิวินาที ในทางกลับกันเทคนิค BS มีประสิทธิภาพเวลาที่เร็วที่สุดในการค้นหาต่อคำหลัก สำหรับกรณีดรรชนีแบบกระจาย โดยใช้เวลาเฉลี่ยเท่ากับ 0.011 มิลลิวินาที และพบว่าเทคนิค LS มีประสิทธิภาพเวลาที่ช้าที่สุดในการค้นหาต่อคำหลักทั้งสองดรรชนี

Title: THE DEVELOPMENT OF INDEXING TECHNIQUES FOR LARGE SCALE DATA IN THE JAVASCRIPT OBJECT FORMAT

Author: Jirawat Duangkaew, Thesis: M.En. (Computer Engineering), University of Phayao, 2023

Advisor: Assistant Professor Dr. Bowonsak Srisungsittisunti

Keywords: Dense Indexing, Sparse Indexing, Large Scale Dataset, Not only Structured Query Language, JavaScript Object Notation

ABSTRACT

The use of JavaScript Object Notation (JSON) in Not only Structured Query Language (NoSQL) has become increasingly popular. However, considering the limitations of NoSQL, particularly the Dense Index technique in a One–to–One format and the Sparse Index in a One–to–Many format for large–scale JSON files, these present significant challenges. This research proposes The development of Indexing techniques for large–scale data in the JavaScript object format. The researcher simulated data of no less than eight hundred thousand entries, with data file sizes of no less than thirty gigabytes, under the JSON files. This data was then subjected to an analytical design process, incorporating some techniques of Relational databases, namely Dense and Sparse Index, Linear Search (LS) techniques, and the application of an Array structure. Binary search (BS), Binary search Tree (BST), and Adelson–Velskii Landis Tree (AVL Tree) search techniques were used, along with Python for performance comparison testing in terms of the time duration of the index in data retrieval. The researcher incorporated these techniques into the process of developing an index for large–scale JSON files, including a method for creating a Dense and a Sparse Index. Subsequently, the tree structure was adjusted to suit the data set of the BS, BST, and AVL Tree techniques, which are under the large–scale JSON files. Ultimately, the research found that the Dense Index approach could reduce data access time by up to 98.57% compared to the Non Index method, and the Sparse Index could reduce data access time by up to 98.45%. Furthermore, when enhancing data access performance per keyword with LS, BS, BST, and AVL Tree techniques. The research found that the AVL Tree technique had the fastest time performance in searching per keyword for the Dense Index case, with an average time of 0.005 milliseconds. Conversely, the BS technique had the fastest time performance in searching per keyword for the Sparse Index case, with an average time of 0.011 milliseconds. The LS technique also demonstrated the slowest time performance when searching per keyword for both index types.

กิตติกรรมประกาศ

วิทยานิพนธ์นี้สำเร็จลุล่วงไปด้วยดีเพราะได้รับความอนุเคราะห์จากท่านผู้ช่วยศาสตราจารย์ ดร.บวรศักดิ์ ศรีสังสิทธิสันติ อาจารย์ที่ปรึกษาวิทยานิพนธ์ ซึ่งท่านเป็นผู้กรุณาให้ความรู้กรอบแนวคิด คำปรึกษาแนะนำ และตรวจสอบแก้ไขจนวิทยานิพนธ์นี้สำเร็จลุล่วงด้วยดี ผู้วิจัยขอกราบขอบพระคุณอาจารย์ที่ปรึกษาวิทยานิพนธ์ เป็นอย่างสูงไว้ ณ โอกาสนี้

ผู้วิจัยขอกราบขอบพระคุณท่านผู้ช่วยศาสตราจารย์ ดร.พรเทพ โรจนวสุ และผู้ช่วยศาสตราจารย์ ดร.สาคร เมฆรักษาวนิช ที่เสียสละเวลาอันมีค่ากรุณามาเป็นคณะกรรมการสอบวิทยานิพนธ์ อีกทั้งให้คำปรึกษาแนะนำ ข้อเสนอความคิดเห็นที่เป็นประโยชน์ ตลอดจนตรวจสอบแก้ไข จนทำให้วิทยานิพนธ์นี้ มีความถูกต้องสมบูรณ์มากยิ่งขึ้น และขอขอบคุณเจ้าหน้าที่หลักสูตรวิศวกรรมศาสตรมหาบัณฑิต สาขาวิศวกรรมคอมพิวเตอร์ สำหรับความเอื้อเฟื้อและความมีน้ำใจในการดำเนินงานทางด้านต่าง ๆ สำเร็จลุล่วงด้วยดีตลอดมา

ท้ายสุดนี้ หากมีข้อผิดพลาดบกพร่องประการใด ผู้วิจัยขออภัยเป็นอย่างสูงไว้ ณ ที่นี้ ผู้วิจัยหวังเป็นอย่างยิ่งว่าวิทยานิพนธ์นี้จะมีประโยชน์สำหรับผู้ที่สนใจ ประสงค์ที่จะศึกษารายละเอียดเกี่ยวกับการพัฒนาเทคนิคสร้างดรรชนีสำหรับข้อมูลขนาดใหญ่ที่อยู่ในรูปแบบวัตถุของจาวาสคริปต์ (The development of indexing techniques for large-scale data in the JavaScript object format)

จิรวัฒน์ ดวงแก้ว

สารบัญ

หน้า

สารบัญตาราง

หน้า

สารบัญภาพ

หน้า

บทที่ 1

บทนำ

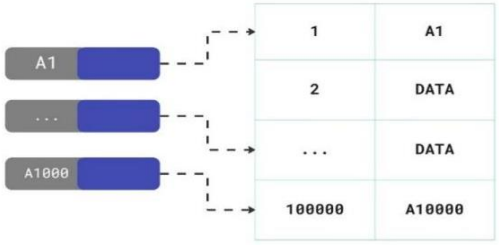
1.1 ความเป็นมาและความสำคัญของปัญหา

ในยุคปัจจุบันที่ปริมาณข้อมูลดิจิทัลเพิ่มขึ้นอย่างต่อเนื่องทุกวันจากระบบอินเทอร์เน็ต ซึ่งข้อมูลเหล่านี้สามารถเกิดขึ้นได้จากการเก็บรวบรวมข้อมูลของตัวอักษร เอกสาร รูปภาพ และข้อมูลดิจิทัลทุกประเภทที่มีอยู่บนอินเทอร์เน็ต อันก่อให้เกิดปัญหาหลายอย่างทางด้านเทคนิค โดยเฉพาะทางด้านการจัดทำดรรชนีข้อมูล (Indexing) เพื่อเพิ่มประสิทธิภาพของการเข้าถึงข้อมูลได้อย่างมีประสิทธิภาพ สามารถแบ่งประเภทของข้อมูลได้ดังนี้

1) ข้อมูลชนิดเลขจำนวนเต็ม (Integer)
2) ข้อมูลชนิดเลขทศนิยม (Float)
3) ข้อมูลชนิดเลขฐานแปด (Octal)
4) ข้อมูลชนิดเลขฐานสิบหก (Hexadecimal)
5) ข้อมูลชนิดตัวอักขระ (Character)
6) ข้อมูลชนิดข้อความ (String)

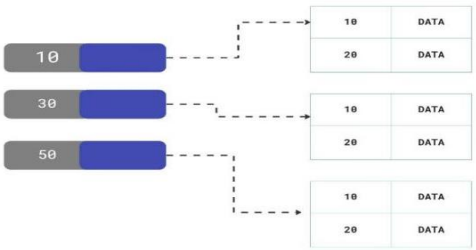
ชุดข้อมูลข้างต้นจะถูกส่งต่อเข้ากระบวนการเก็บรวบรวมข้อมูลเป็นศูนย์กลางข้อมูลอย่างเป็นระบบ เช่น ฐานข้อมูลเชิงสัมพันธ์ (Relational Database) และฐานข้อมูลไม่ใช่เชิงสัมพันธ์ (Not only Structured Query Language, NoSQL) เมื่อมีการสะสมของข้อมูลอย่างต่อเนื่องของทุกวัน จึงต้องนำเทคนิคการจัดทำดรรชนี เข้ามาช่วยในการสืบค้นหาข้อมูลที่ร้องขออย่างถูกต้อง แม่นยำ รวดเร็ว และมีประสิทธิภาพ ซึ่งดรรชนีของฐานข้อมูลเชิงสัมพันธ์เป็นวิธีหนึ่งที่สามารถเพิ่มประสิทธิภาพในการลดจำนวนการเข้าถึงฮาร์ดดิสก์ (Hard disk) ที่จำเป็นเมื่อประมวลผลการสืบค้นข้อมูล ซึ่งเป็นเทคนิคทางโครงสร้างข้อมูลที่ใช้ในการค้นหาและเข้าถึงฐานข้อมูลได้อย่างรวดเร็ว โดยเทคนิคพื้นฐานการจัดทำดรรชนีของฐานข้อมูลเชิงสัมพันธ์ ต้องคำนึงถึงวิธีการเก็บข้อมูล การจัดระเบียบไฟล์ ซึ่งสามารถแบ่งประเภทดังนี้

1) การจัดทำดรรชนีแบบหนาแน่น (Dense Indexing)



ภาพ 1 ดรรชนีแบบหนาแน่น (Dense Indexing)

2) การจัดทำดรรชนีที่ไม่เป็นกลุ่ม (ดรรชนีที่ไม่ใช่คลัสเตอร์, Non-clustered) หรือดรรชนีรอง (Non-clustered or Secondary Indexing)



ภาพ 2 ดรรชนีรอง (Non-clustered or Secondary Indexing)

การจัดทำดรรชนี (Indexing) สำหรับฐานข้อมูลเชิงสัมพันธ์ (Relational database) ยังคงมีข้อจำกัดที่สำคัญในด้านของเวลา โดยเฉพาะในขั้นตอนการสแกนข้อมูลรายการชุดฐานข้อมูลอีกครั้งก่อนเริ่มกระบวนการจัดทำดรรชนีข้อมูล ในสภาวะที่ปริมาณข้อมูลมีแนวโน้มขยายการเติบโตอย่างต่อเนื่อง ความล่าช้าในระยะเวลาในการสืบค้นข้อมูลจึงเป็นเรื่องที่ไม่สามารถหลีกเลี่ยงได้ นอกจากนี้ ประสิทธิภาพของข้อมูลที่ได้จากกระบวนการดังกล่าวยังมีโอกาสเกิดข้อผิดพลาดจากโครงสร้างข้อมูล (Dittrich, Nix และSchön, 2021; GeeksforGeeks, n.d.; IBM, n.d.)

ช่วงคริสต์ศักราช Carlo Strozzi (1998) พัฒนาฐานข้อมูลไม่ใช่เชิงสัมพันธ์ (Not only Structured Query Language, NoSQL) ซึ่งมีคุณสมบัติ คือไม่ต้องจัดทำสคีมา (No–Schema) ฐานข้อมูลสามารถปรับขนาดข้อมูลได้ในแนวนอน (Horizontally scalable) สถาปัตยกรรมดรรชนีแบบกระจายและทนต่อความผิดพลาดรวมถึงละเว้นการฏิบัติตามของหลักการ (ACID: Atomicity, Consistency, Isolation, Durability) สำหรับประเภทฐานข้อมูลไม่ใช่เชิงสัมพันธ์ (Not only Structured Query Language, NoSQL) ประกอบด้วยสี่ประเภทหลักดังนี้

1) ฐานข้อมูลประเภทเชิงเอกสาร (Document Databases)
2) ฐานข้อมูลประเภทคีย์หรือค่าของข้อมูล (Key/Value Databases)
3) ฐานข้อมูลประเภทตระกูลคอลัมน์ (Column Family Databases)
4) ฐานข้อมูลประเภทกราฟ (Graph Databases)

สำหรับฐานข้อมูลไม่ใช่เชิงสัมพันธ์ (Not only Structured Query Language, NoSQL) เทคนิคการจัดทำดรรชนี (Indexing) ปัจจุบันยังคงมีข้อจำกัดที่ท้าทาย เนื่องจากเป็นภาษาที่เกิดขึ้นตั้งแต่ช่วงคริสต์ศักราช 1998 ทำให้งานวิจัยยังไม่แพร่หลายที่เกี่ยวข้องกับการจัดทำดรรชนี สำหรับฐานข้อมูลที่ไม่มีโครงสร้างเชิงสัมพันธ์ โดยมุ่งเน้นปรับปรุงเพิ่มประสิทธิภาพให้กับดรรชนีของข้อมูล รวมถึงความท้าทายที่จะไม่ต้องผ่านกระบวนแสกนข้อมูลชุดรายการฐานของข้อมูลใหม่อีกครั้งกับฐานข้อมูลขนาดใหญ่ (Elmasri และNavathe, 2011; Han และคณะ, 2011)

เมื่อพิจารณาถึงคุณสมบัติที่ดีและข้อจำกัด พบว่าฐานข้อมูลเชิงสัมพันธ์ (Relational Database) มีคุณสมบัติที่ดี คือ รองรับการจัดทำหมวดหมู่เพื่อจัดเก็บข้อมูลในรูปแบบฐานข้อมูลเชิงสัมพันธ์ ทำให้เมื่อมีการร้องขอข้อมูลนั้นมีความแม่นยำถูกต้อง ลดความซับซ้อนในการจัดการกับข้อมูล อันเนื่องจากการกำหนดความสัมพันธ์ของฐานข้อมูล อีกทั้งมีความปลอดภัยจากการกำหนดสิทธิ์เข้าถึงของข้อมูล ส่วนทางด้านข้อจำกัดของฐานข้อมูลเชิงสัมพันธ์ คือค่าใช้จ่ายสูง เพราะเนื่องจากต้องการทรัพยากรทางด้านฮาร์ดแวร์ ซอฟต์แวร์ ที่เหมาะสมกับระบบฐานข้อมูลแล้ว ยังต้องมีผู้เชี่ยวชาญทางเทคนิคเฉพาะด้านเข้ามาปรับปรุงและพัฒนาฐานข้อมูลเชิงสัมพันธ์ให้เป็นไปตามความต้องการของระบบ อีกทั้งเมื่อเวลาผ่านไปปริมาณจำนวนข้อมูลสะสมเพิ่มขึ้นอย่างต่อเนื่อง ส่งผลกระทบต่อประสิทธิภาพลดลงในหลายมิติจนถึงความซับซ้อนของโครงสร้างฐานข้อมูล มีโอกาสเกิดข้อผิดพลาดของข้อมูลได้ สำหรับฐานข้อมูลไม่ใช่เชิงสัมพันธ์ (Not only Structured Query Language, NoSQL) มีคุณสมบัติที่ดีคือไม่จำเป็นต้องมีโครงสร้างของฐานข้อมูลเชิงสัมพันธ์ สามารถปรับขนาดในแนวนอน รองรับความยืดหยุ่นในการปรับขนาดได้สูง อีกทั้งสามารถจัดเก็บและดึงข้อมูล โดยไม่ต้องจัดทำสคีมา

(No-Schema) ที่กำหนดไว้ล่วงหน้าอีกทั้งง่ายต่อการบำรุงรักษาฐานข้อมูล เนื่องจากสามารถกำหนดแบ่งพาร์ทิชัน และจำลองข้อมูลข้ามโหนดได้โดยอัตโนมัติ ส่วนทางด้านข้อจำกัดของฐานข้อมูลที่ไม่มีโครงสร้างเชิงสัมพันธ์ คือ การสืบค้นหาข้อมูลมีความยืดหยุ่นน้อยกว่าฐานข้อมูลเชิงสัมพันธ์ (Relational Database) เนื่องจากฐานข้อมูลที่ไม่มีโครงสร้างเชิงสัมพันธ์มีความยืดหยุ่นทางโครงสร้างฐานของข้อมูล อย่างไรก็ตามยังไม่รองรับฟังก์ชันการสืบค้นหาข้อมูลที่ซับซ้อน ส่งผลให้ยากต่อการทำดรรชนี (Indexing) อีกทั้งยากต่อรองรับกระบวนการของ ACID (Atomicity, Consistency, Isolation, Durability) (Elmasri และNavathe, 2011; MongoDB, n.d.) เมื่อพิจารณาถึงคุณสมบัติ ข้อดี ข้อจำกัดของฐานข้อมูลเชิงสัมพันธ์ (Relational Database) และฐานข้อมูลไม่ใช่เชิงสัมพันธ์ (Not only Structured Query Language, NoSQL) พบว่ามีความท้าทายเป็นอย่างมากสำหรับการจัดทำดรรชนี (Indexing) บนฐานข้อมูลที่ไม่มีโครงสร้างเชิงสัมพันธ์ งานวิจัยมุ่งเน้นทางด้านการเพิ่มประสิทธิภาพให้การสืบค้นข้อมูลได้เร็วยิ่งขึ้น ดรรชนีจะไม่แสกนข้อมูลชุดรายการฐานข้อมูลใหม่อีกครั้งกับลักษณะดรรชนีแบบหนาแน่น (Dense Index) แบบหนึ่งต่อหนึ่ง (One-to-One) และดรรชนีแบบกระจาย (Sparse Index) แบบหนึ่งต่อกลุ่ม (One-to-Many) สำหรับชุดข้อมูลไฟล์ JSON ขนาดใหญ่ งานวิจัยครั้งนี้ได้เสนอการพัฒนาเทคนิคสร้างดรรชนีสำหรับข้อมูลขนาดใหญ่ที่อยู่ในรูปแบบวัตถุของจาวาสคริปต์ (JavaScript Object Notation, JSON) โดยนำข้อมูลไฟล์ JSON ไม่ต่ำกว่าแปดแสนรายการ มีขนาดไฟล์ไม่ต่ำกว่าสามสิบกิกะไบต์ เข้าสู่กระบวนการวิเคราะห์ออกแบบขั้นตอนวิธีซึ่งผสมผสานบางเทคนิคของฐานข้อมูลเชิงสัมพันธ์ (Relational Database) ได้แก่ การจัดทำดรรชนีแบบหนาแน่น (Dense Index) และดรรชนีแบบกระจาย (Sparse Index) และการประยุกต์ใช้โครงสร้างแบบแถวลำดับ (Array) เทคนิคการค้นหาเชิงเส้น (Linear Search, LS) เทคนิคการค้นหาแบบทวิภาค (Binary Search, BS) เทคนิคต้นไม้ค้นหาแบบทวิภาค (Binary Search Tree, BST) และ เทคนิค Adelson-Velskii Landis Tree (AVL Tree) รวมถึงนำภาษาไพทอน (Python) สำหรับทดสอบเปรียบเทียบประสิทธิภาพทางด้านเวลาของดรรชนีในการสืบค้นหาข้อมูล เทคนิคข้างต้นนำเข้าสู่กระบวนการพัฒนาเทคนิคสร้างดรรชนีสำหรับข้อมูลขนาดใหญ่ที่อยู่ในรูปแบบวัตถุของจาวาสคริปต์ ได้แก่ ขั้นตอนวิธีสำหรับการจัดทำดรรชนีแบบหนาแน่นและดรรชนีแบบกระจาย จากนั้นดำเนินการปรับแต่งโครงสร้างต้นไม้ให้เหมาะสมกับข้อมูลด้วยเทคนิค BS, BST และ AVL Tree สำหรับชุดข้อมูล JSON ขนาดใหญ่

1.2 สมมติฐานวิจัย

การจัดทำดรรชนี (Indexing) ที่ปรับปรุงพัฒนาใหม่ สามารถเพิ่มประสิทธิภาพให้การสืบค้นข้อมูลได้เร็วขึ้นของลักษณะดรรชนีแบบหนาแน่นแบบหนึ่งต่อหนึ่ง (Dense Index) และดรรชนีแบบกระจายแบบหนึ่งต่อกลุ่ม (Sparse Index) สำหรับชุดข้อมูลไฟล์ JSON ขนาดใหญ่

1.3 วัตถุประสงค์ของการวิจัย

1.3.1 เพื่อเพิ่มประสิทธิภาพของดรรชนี (Indexing) ทางด้านการสำรวจและเข้าถึงข้อมูลได้เร็วขึ้นกับดรรชนีแบบหนาแน่นแบบหนึ่งต่อหนึ่ง (Dense Index) สำหรับชุดข้อมูลไฟล์ JSON ขนาดใหญ่

1.3.2 เพื่อเพิ่มประสิทธิภาพของดรรชนี (Indexing) ทางด้านการสำรวจและเข้าถึงข้อมูลได้เร็วขึ้นกับดรรชนีแบบกระจายแบบหนึ่งต่อกลุ่ม (Sparse Index) สำหรับชุดข้อมูลไฟล์ JSON ขนาดใหญ่

1.4 ขอบเขตของการวิจัย

1.4.1 เพื่อเพิ่มประสิทธิภาพของดรรชนี (Indexing) ทางด้านการสืบค้นข้อมูลได้เร็วขึ้นกับลักษณะลักษณะดรรชนีแบบหนาแน่นแบบหนึ่งต่อหนึ่ง (Dense Index) และดรรชนีแบบกระจายแบบหนึ่งต่อกลุ่ม (Sparse Index) สำหรับชุดข้อมูลไฟล์ JSON ขนาดใหญ่

1.4.2 จำลองชุดรายการข้อมูลขนาดใหญ่ไม่ต่ำกว่าแปดแสนรายการขึ้นไป และมีขนาดไฟล์ไม่ต่ำกว่าสามสิบกิ๊กกะไบต์ขึ้นไป สำหรับชุดข้อมูลไฟล์ JSON ขนาดใหญ่

1.4.3 ทดสอบเปรียบเทียบประสิทธิภาพของดรรชนี (Indexing) ทางด้านการสืบค้นข้อมูลได้เร็วขึ้นกับลักษณะดรรชนีแบบหนาแน่นแบบหนึ่งต่อหนึ่ง (Dense Index) และดรรชนีแบบกระจายแบบหนึ่งต่อกลุ่ม (Sparse Index) ชุดข้อมูลไฟล์ JSON ขนาดใหญ่

1.5 ประโยชน์ของการวิจัย

1.5.1 สามารถเพิ่มประสิทธิภาพของดรรชนี (Indexing) ทางด้านการสืบค้นข้อมูลได้เร็วขึ้นกับลักษณะดรรชนีแบบหนาแน่นแบบหนึ่งต่อหนึ่ง (Dense Index) และดรรชนีแบบกระจายแบบหนึ่งต่อกลุ่ม (Sparse Index) สำหรับชุดข้อมูลไฟล์ JSON ขนาดใหญ่

1.5.2 สามารถจำลองชุดรายการข้อมูลขนาดใหญ่ไม่ต่ำกว่าแปดแสนรายการขึ้นไป และมีขนาดไฟล์ไม่ต่ำกว่าสามสิบกิ๊กกะไบต์ขึ้นไป สำหรับชุดข้อมูลไฟล์ JSON ขนาดใหญ่

1.5.3 สามารถทดสอบเปรียบเทียบประสิทธิภาพของดรรชนี (Indexing) ทางด้านการสืบค้นข้อมูลได้เร็วขึ้นกับลักษณะดรรชนีแบบหนาแน่นแบบหนึ่งต่อหนึ่ง (Dense Index) ดรรชนีแบบกระจายแบบหนึ่งต่อกลุ่ม (Sparse Index) สำหรับชุดข้อมูลไฟล์ JSON ขนาดใหญ่

1.6 ขั้นตอนของการดำเนินงานวิจัย

1.6.1 ศึกษาทฤษฎีและงานวิจัยที่เกียวข้องทางด้านการจัดทำดรรชีนีของฐานข้อมูล

1.6.2 วิเคราะห์และออกแบบ

1.6.3 พัฒนาอัลกอริทึมตามที่ได้ออกแบบไว้ โดยใช้ภาษาไพทอน (Python)

1.6.4 จำลองสำหรับชุดข้อมูลไฟล์ JSON ขนาดใหญ่ ตามที่อธิบายไว้ใน 3.1 รวมถึงการคัดเลือกคำหลักที่มีลักษณะการแจกแจงทั้งสามรูปแบบของดรรชนีแบบหนาแน่น (Dense Index) และดรรชนีแบบกระจาย (Sparse Index) ตามที่อธิบายไว้ใน 3.4 เพื่อใช้สำหรับการทดลองเปรียบเทียบประสิทธิภาพการเข้าถึงข้อมูลด้วยดรรชนี

1.6.5 สรุปผลการทดลอง

1.6.6 จัดทำรายงานการวิจัย

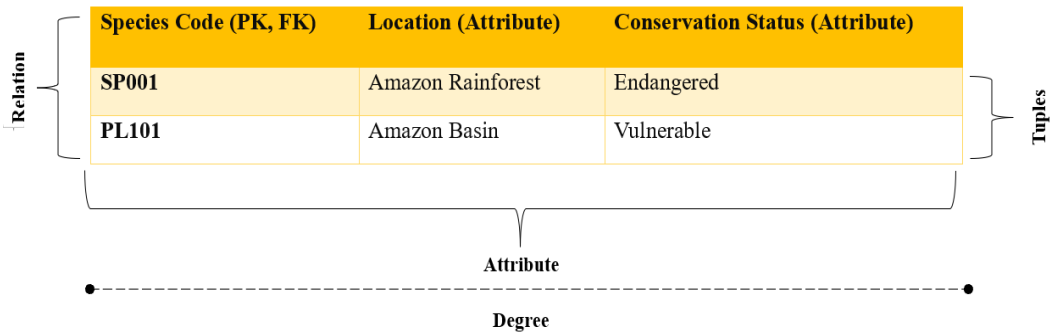
บทที่ 2

ทฤษฎี และงานวิจัยที่เกี่ยวข้อง

งานวิจัยนี้ได้นำเสนอการพัฒนาเทคนิคสร้างดรรชนีสำหรับข้อมูลขนาดใหญ่ที่อยู่ในรูปแบบวัตถุของจาวาสคริปต์ (The development of indexing techniques for large–scale data in the JavaScript object format) ผู้วิจัยได้ศึกษา ทฤษฎี และงานวิจัยที่เกี่ยวข้องซึ่งประกอบด้วยดังนี้

2.1 ทฤษฎีสำหรับฐานข้อมูลเชิงสัมพันธ์

(Elmasri และNavathe, 2011) ฐานข้อมูลเชิงสัมพันธ์ (Relational Database) พัฒนาโดย Dr. Edgar Frank Codd จาก International Business Machines Research (IBM Research) ในปีคริสต์ศักราช 1960 ซึ่งมีรากฐานมาจากหลักทฤษฎีเซต (Set theory) และทฤษฎีตรรกะเชิงนิพจน์ (Predicate logic theory) ทฤษฎีเหล่านี้นำไปสู่การพัฒนาแนวคิดในการออกแบบฐานข้อมูลเชิงสัมพันธ์สำหรับการจัดการกับข้อมูลขนาดใหญ่ ถูกประยุกต์ใช้เชิงพาณิชย์ครั้งแรกบนระบบ Structured Query Language/Data System (SQL/DS) โดย International Business Machines (IBM) และ Oracle Corporation ตอนต้นทศวรรษ 1980 ซึ่งหลังจากนั้นได้แพร่กระจายไปยังระบบโอเพนซอร์สที่นิยมอย่าง MySQL และ PostgreSQL ที่ใช้แบบจำลองนี้อย่างกว้างขวาง โครงสร้างความสัมพันธ์ของฐานข้อมูลประเภทนี้ประกอบด้วยตารางที่มีแถว (Row) และคอลัมน์ (Column) เพื่อบ่งบอกคุณลักษณะ (Attribute) ของข้อมูล ทูเปิล (Tuple) ในฐานข้อมูลนี้จะประกอบด้วยคุณลักษณะจำนวนที่แน่นอนในตาราง ดีกรี (Degree) ระบุระดับของจำนวนตารางที่เชื่อมต่อกัน แสดงถึงความสัมพันธ์ระหว่างตารางต่างๆ คีย์หลัก (Primary key) ในแต่ละตารางคือตัวระบุเฉพาะสำหรับแต่ละระเบียน ในขณะที่คีย์รอง (Foreign key) คือการอ้างอิงถึงคีย์หลักของตารางอื่น เพื่อรับรองความถูกต้องของข้อมูลและบังคับใช้ความสมบูรณ์ของการอ้างอิง โดเมน (Domain) คือชุดของค่าที่สามารถจัดเก็บในคอลัมน์ ช่วยให้มั่นใจในความถูกต้องของข้อมูล การกำหนดโดเมนช่วยจำกัดค่าที่สามารถจัดเก็บในคอลัมน์ได้ การประยุกต์ใช้งานของแนวคิดฐานข้อมูลเชิงสัมพันธ์ในการจัดการข้อมูลที่มีขนาดใหญ่ ดังแสดงตัวอย่างภาพ 3 ข้อมูลของป่าอเมซอน ซึ่งสะท้อนถึงความสำคัญของแนวคิดนี้ของการจัดเก็บข้อมูลรวมถึงการพัฒนาทางด้านการจัดการข้อมูลขนาดใหญ่และการวิเคราะห์ข้อมูล



ภาพ 3 สถาปัตยกรรมของตารางฐานข้อมูลเชิงสัมพันธ์

2.2 ทฤษฎีสำหรับฐานข้อมูลที่ไม่ใช่เชิงสัมพันธ์

(Meier และคณะ, 2019) ฐานข้อมูลที่ไม่ใช่เชิงสัมพันธ์ (Not only Structured Query Language, NoSQL) ถูกคิดค้นขึ้นโดย Carlo Strozzi ในช่วงปีคริสต์ศักราช 1998 ซึ่งเป็นหมวดหมู่ของฐานข้อมูลซึ่งไม่ยึดตามโครงสร้างทั่วไปของฐานข้อมูลเชิงสัมพันธ์ (Relational Database) ฐานข้อมูลเหล่านี้มีความยืดหยุ่นในการจัดเก็บข้อมูลสามารถปรับเปลี่ยนและปรับขนาดได้มากกว่าฐานข้อมูลเชิงสัมพันธ์ซึ่งจัดระเบียบข้อมูลโดยใช้ตาราง (Table) และแถว (Row) องค์ประกอบหลักของฐานข้อมูลที่ไม่มีโครงสร้างเชิงสัมพันธ์นี้สามารถจำแนกออกได้เป็นสี่ประเภทหลักดังนี้

1) ฐานข้อมูลประเภทเชิงเอกสาร (Document Storage) เป็นวิธีการจัดเก็บข้อมูลที่ใช้งานเอกสารดิจิทัลในรูปแบบไบนารี เช่น Binary JSON (BSON) หรือ Concise Binary Object Representation (CBOR) ภายในหน่วยการจัดเก็บหลักที่เรียกว่า การเก็บรวบรวม (Collections) ซึ่งภายในหน่วยนี้ แต่ละเอกสารสามารถมีโครงสร้างที่แตกต่างกัน ความยืดหยุ่นดังกล่าวช่วยให้สามารถปรับขนาดและปรับเปลี่ยนโครงสร้างข้อมูลได้สะดวก เพื่อรองรับความหลากหลายและความซับซ้อนของข้อมูล นอกจากนี้ ฐานข้อมูลเชิงเอกสารช่วยลดความซับซ้อนในการจัดการข้อมูลเมื่อเปรียบเทียบกับฐานข้อมูลเชิงสัมพันธ์ที่มีข้อจำกัดด้านโครงสร้างและความสัมพันธ์ของข้อมูล ซึ่งเหมาะสมสำหรับการใช้งานในสภาพแวดล้อมที่ต้องการความยืดหยุ่นสูงและการปรับเปลี่ยนข้อมูลอย่างรวดเร็ว

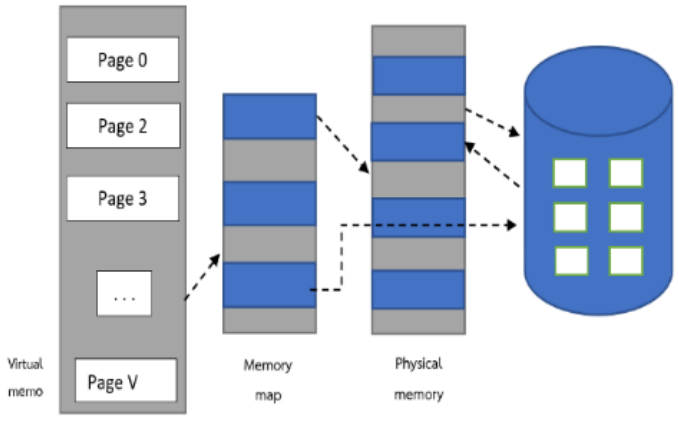
2) ฐานข้อมูลประเภทคีย์หรือค่าของข้อมูล (Key/Value Storage) เป็นเทคนิคการจัดเก็บข้อมูลโดยใช้คีย์ (Key) เป็นตัวระบุข้อมูลแต่ละส่วน และค่า (Value) เป็นข้อมูลที่จะจัดเก็บ คีย์นั้นเป็นตัวระบุที่ชัดเจนและไม่ซ้ำใครสำหรับแต่ละส่วนของข้อมูล โดยที่ค่าสามารถเป็นข้อมูลใดๆ ก็ตามที่เกี่ยวข้องกับคีย์นั้น การจัดเก็บข้อมูลแบบนี้ช่วยให้การค้นหาและการจัดการข้อมูลมีประสิทธิภาพสูง เพราะสามารถเข้าถึงข้อมูลที่ต้องการได้โดยตรงผ่านคีย์ ซึ่งต่างจากฐานข้อมูลเชิงสัมพันธ์ที่มีความซับซ้อนในการจัดการและเชื่อมโยงข้อมูล ฐานข้อมูลประเภทนี้เหมาะกับการใช้งานที่ต้องการการเข้าถึงข้อมูลอย่างรวดเร็วและโครงสร้างข้อมูลที่ไม่ซับซ้อน นอกจากนี้ยังรองรับการประมวลผลข้อมูลขนาดใหญ่และการจัดเก็บข้อมูลที่หลากหลายได้อย่างมีประสิทธิภาพ.

3) ฐานข้อมูลประเภทตระกูลคอลัมน์ (Column Family Storage) เป็นเทคนิคการจัดเก็บข้อมูลที่ใช้วิธีการเก็บข้อมูลตามตระกูลคอลัมน์ แทนที่การใช้วิธีการจัดเก็บแบบแถว (Row Storage) ซึ่งเป็นวิธีทั่วไปในฐานข้อมูลเชิงสัมพันธ์ (Relational Database) ตระกูลคอลัมน์เป็นแบบจำลองข้อมูลที่จัดกลุ่มข้อมูลตามคอลัมน์และใช้ดรรชนีแบบกระจาย (Distributed Index) เพื่อกระจายข้อมูลไปยังโหนดต่างๆ ในระบบ ทำให้การจัดการข้อมูล การสืบค้น และการดึงข้อมูลเป็นไปอย่างมีความยืดหยุ่นและมีประสิทธิภาพ ฐานข้อมูลประเภทนี้เหมาะกับแอปพลิเคชันที่ต้องการปรับขนาดได้และมีประสิทธิภาพสูง ทำให้เป็นทางเลือกที่ดีสำหรับการจัดเก็บข้อมูลขนาดใหญ่และการเข้าถึงข้อมูลที่ซับซ้อน นอกจากนี้ยังรองรับการจัดเก็บข้อมูลที่มีโครงสร้างแบบไดนามิกและเปลี่ยนแปลงได้ตามความต้องการ.

4) ฐานข้อมูลประเภทกราฟ (Graph Storage) เป็นเทคนิคการจัดเก็บข้อมูลที่ใช้โครงสร้างข้อมูลแบบกราฟในการจัดเก็บและจัดการข้อมูล ในโครงสร้างกราฟ ข้อมูลถูกแสดงออกในรูปของโหนด (Node) และขอบ (Edge) โดยโหนดแทนสิ่งต่างๆ และเอดจ์แทนความสัมพันธ์ระหว่างโหนดเหล่านั้น ความสามารถในการเชื่อมโยงข้อมูลแบบซับซ้อนและสะท้อนโครงสร้างและความสัมพันธ์ที่เป็นธรรมชาติของข้อมูลทำให้ฐานข้อมูลประเภทกราฟเหมาะกับการประมวลผลข้อมูลที่เน้นความสัมพันธ์ อีกทั้งยังมีประสิทธิภาพในการวิเคราะห์เครือข่ายที่ซับซ้อนและการจัดการข้อมูลในรูปแบบที่เชื่อมโยงกันอย่างหลากหลาย ซึ่งเหนือกว่าฐานข้อมูลเชิงสัมพันธ์ในหลายด้าน นอกจากนี้ ยังเหมาะสำหรับการจัดเก็บข้อมูลที่มีความซับซ้อนและต้องการวิเคราะห์ความสัมพันธ์ระหว่างข้อมูลในรูปแบบแตกต่างกัน ทั้งยังเสริมสร้างความสามารถในการทำความเข้าใจภาพรวมและลึกซึ้งไปยังเครือข่ายข้อมูลที่ซับซ้อนได้อย่างมีประสิทธิภาพ.

2.3 ทฤษฎีหน่วยความจำเสมือน

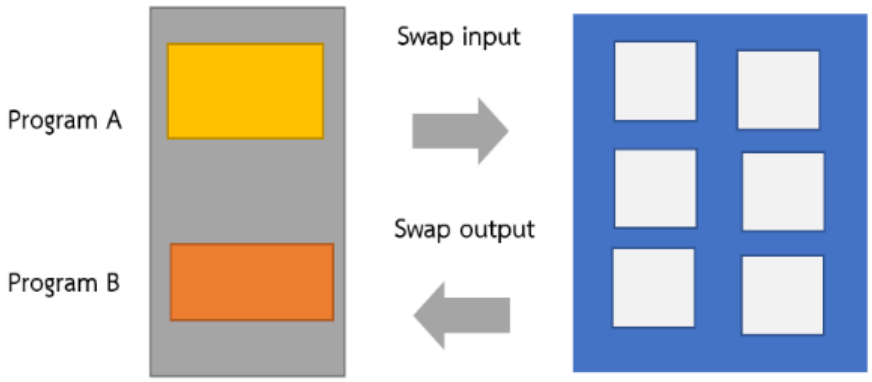
(Silberschatz, 2013) ทฤษฎีหน่วยความจำเสมือน (Virtual Memory) มีบทบาทสำคัญในการขยายขีดจำกัดของหน่วยความจำหลัก (Random Access Memory, RAM) ในระบบปฏิบัติการ (Operating System, OS) ด้วยการนำเสนอการใช้พื้นที่จัดเก็บของอุปกรณ์จัดเก็บข้อมูลรอง เช่น ฮาร์ดดิสก์ (Hard Disk) หรือโซลิดสเตตไดรฟ์ (Solid State Drive, SSD) ในฐานะหน่วยความจำเสมือน ซึ่งทำหน้าที่เสมือนหน่วยความจำหลักเมื่อ RAM ไม่เพียงพอต่อความต้องการ การดำเนินการนี้ช่วยให้โปรแกรมสามารถดำเนินการได้โดยไม่สะดุดแม้ในสภาวะที่หน่วยความจำหลักเต็ม อย่างไรก็ตาม การใช้งานหน่วยความจำเสมือนอาจนำไปสู่การลดลงของประสิทธิภาพเนื่องจากความล่าช้าในการเข้าถึงข้อมูลจากพื้นที่จัดเก็บข้อมูลรองที่มีความเร็วน้อยกว่าหน่วยความจำหลัก การอัพเกรดเป็น SSD สามารถลดปัญหานี้ได้ เนื่องจาก SSD มีความเร็วในการอ่านและเขียนข้อมูลที่สูงกว่า ซึ่งช่วยเพิ่มประสิทธิภาพในการเข้าถึงข้อมูลและประสิทธิภาพการทำงานของ OS การอัพเกรดหน่วยความจำหลักยังเป็นวิธีที่มีประสิทธิภาพในการขยายความสามารถของระบบปฏิบัติการ ช่วยลดการพึ่งพาหน่วยความจำเสมือน และเสริมความเสถียรในการทำงานของแอปพลิเคชัน



ภาพ 4 หน่วยความจำเสมือน (Virtual memory)

โดยทั่วไปหลักการสำคัญของหน่วยความจำเสมือน ได้แก่ การจัดสรรหน้าตามคำร้องขอ (Demand Paging) ซึ่งมีรายละเอียดดังต่อไปนี้

2.3.1 การจัดสรรหน้าตามคำร้องขอ (Demand Paging) คือ เทคนิคในการจัดการหน่วยความจำ โดยการโอนย้ายข้อมูลจากหน่วยความจำรอง (Secondary Memory) เข้าสู่หน่วยความจำหลัก (Random Access Memory, RAM) แบบเลือกตามความจำเป็นภายใต้การจำกัดพื้นที่ของระบบปฏิบัติการ (Operating System, OS) ในกรณีที่หน่วยประมวลผลกลาง (Central Processing Unit, CPU) ต้องการเข้าถึงข้อมูลที่ไม่อยู่ใน RAM ระบบจะทำการสลับหน้าข้อมูลที่มีอยู่ใน RAM ด้วยข้อมูลจากหน่วยความจำรอง สร้างความยืดหยุ่นและประสิทธิภาพในการจัดการหน่วยความจำ



ภาพ 5 การจัดสรรหน้าตามคำร้องขอ

2.3.2 การเปลี่ยนหน้า (Page Replacement) เป็นกระบวนการหลักในระบบปฏิบัติการ (Operating System, OS) ที่มีหน้าที่จัดการหน่วยความจำกายภาพ (Physical Memory) และอุปกรณ์เก็บข้อมูล (Storage Devices) อาทิ ฮาร์ดดิสก์ไดรฟ์ (Hard Disk Drive, HDD) ด้วยการแบ่งพื้นที่เก็บข้อมูลเป็นหน้าขนาดเล็ก (Page) และทำการสับเปลี่ยนหน้าที่ไม่ได้ใช้งาน การนำหน้าข้อมูลใหม่เข้าสู่หน่วยความจำ (Memory) ช่วยลดความล่าช้าในการเข้าถึงข้อมูลและเพิ่มประสิทธิภาพของระบบ จุดเด่นคือการเพิ่มความเร็วและประสิทธิภาพในการเข้าถึงข้อมูล แต่มีข้อจำกัดในความซับซ้อนของการจัดการ (Management Complexity) และเวลาที่ใช้ในการสับเปลี่ยน (Swapping Time) ซึ่งอาจมีผลต่อประสิทธิภาพการประมวลผล

1) ขั้นตอนการเข้าก่อนออกก่อน (First In First Out Algorithm, FIFO) เป็นวิธีการจัดการหน้าเพจในหน่วยความจำภายในระบบปฏิบัติการ โดยใช้หลักการคิว โดยหน้าเพจที่เข้ามาก่อนจะถูกเก็บไว้หน้าสุดของคิว ส่วนหน้าใหม่จะถูกเพิ่มไว้ท้ายคิว เมื่อต้องแทนที่หน้าเพจ ระบบจะเลือกหน้าสุดของคิวเป็นหน้าแรกที่ถูกแทนที่ จากนั้นหน้าใหม่จะถูกเพิ่มเข้าท้ายคิว ในขณะที่หน้าเก่าที่สุดจะถูกลบออก เพื่อรักษาสมดุลระหว่างหน้าที่มีอยู่และหน้าใหม่

2) ขั้นตอนการแทนที่หน้าที่เหมาะสม (Optimal Page Replacement Algorithm) เป็นวิธีการเลือกหน้าในหน่วยความจำที่ไม่ได้ใช้งานนานที่สุด เพื่อทำการเปลี่ยนแปลงแทนที่ ซึ่งช่วยลดปัญหาข้อบกพร่องของหน้าได้ การใช้วิธีนี้จะทำให้การเข้าถึงข้อมูลมีประสิทธิภาพ ลดเวลาในการประมวลผล และเพิ่มความเร็วในการทำงานของระบบคอมพิวเตอร์

3) ขั้นตอนการสับเปลี่ยนแบบใช้งานน้อยที่สุด–ออกก่อน (Least Recently Used Page Replacement Algorithm, LRU) เป็นวิธีการจัดการหน่วยความจำเสมือน โดยใช้ประวัติการใช้งานหน้าเพจมาประกอบการตัดสินใจ เมื่อต้องการพื้นที่ใหม่ หน้าที่ไม่ได้ใช้งานนานที่สุดจะถูกเลือกออก วิธีนี้ช่วยเพิ่มประสิทธิภาพการจัดการหน่วยความจำ ลดเวลาตอบสนอง เพิ่มความเร็วในการประมวลผลข้อมูล และทำให้ระบบปฏิบัติการทำงานได้ดีขึ้น

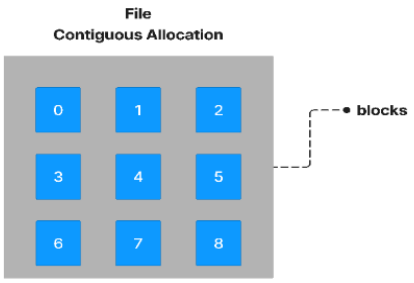
4) ขั้นตอนวิธีการเปลี่ยนหน้าเข้าก่อนออกก่อน (Last In First Out Page Replacement Algorithm, LIFO) เป็นวิธีการจัดการหน่วยความจำในคอมพิวเตอร์ โดยหลักการคือหน้าเว็บใหม่ล่าสุดจะถูกเก็บไว้ด้านบนสุดของ Stack และเมื่อหน่วยความจำเต็ม หน้าด้านบนสุดจะถูกนำออกก่อน วิธีนี้ช่วยให้การจัดการหน่วยความจำมีประสิทธิภาพ แต่มีข้อจำกัดด้านประสิทธิภาพการเข้าถึงข้อมูล เนื่องจากอาจมีการนำหน้าที่มีการใช้งานบ่อยออกไปโดยไม่คำนึงถึงความถี่การใช้งาน

5) ขั้นตอนวิธีการเปลี่ยนหน้าแบบสุ่ม (Random Page Replacement Algorithm) เป็นวิธีการจัดการหน่วยความจำโดยทำการเลือกหน้าที่ต้องการเปลี่ยนออกจากหน่วยความจำแบบสุ่ม โดยไม่สนใจลำดับหรือความถี่การใช้งาน ข้อดีคือมีความเรียบง่ายในการประมวลผล เนื่องจากไม่ต้องเก็บข้อมูลเพิ่มเติมเกี่ยวกับหน้าในหน่วยความจำ อย่างไรก็ตาม ข้อจำกัดคืออาจเปลี่ยนหน้าที่ใช้งานบ่อยออกไปได้ ทำให้อัตราข้อผิดพลาดเพิ่มสูงขึ้น

2.4 ทฤษฎีสำหรับการจัดสรรไฟล์ในฮาร์ดไดรฟ์

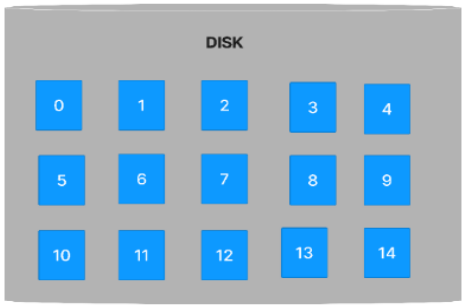
(Silberschatz, 2013) ทฤษฎีสำหรับการจัดสรรไฟล์ในฮาร์ดไดรฟ์ (Files Allocation in hard drive) เป็นกระบวนการใช้งานพื้นที่ของฮาร์ดดิสก์ (Hard Disk) อย่างมีประสิทธิภาพโดยมีหลักการสำคัญหลายประการ เช่น การจัดสรรอย่างต่อเนื่อง (Contiguous Allocation) การจัดสรรที่เชื่อมโยง (Linked Allocation) การจัดทำดรรชนี (Indexed Allocation) และมีแนวคิดที่ควรพิจารณาก่อนการจัดสรรพื้นที่ให้กับไฟล์ อาทิ ปัจจัยทางด้านความเร็วส่วนของการประมวลผลที่มีการเข้าถึงข้อมูลแบบสุ่มลำดับ การถ่ายโอนหลายส่วนข้อมูลและหลายแทร็กข้อมูล (Multi-part Data and Multi-track Data Transfer) ความต้องการใช้พื้นที่ และความต้องการใช้พื้นที่หน่วยความจำหลัก (Main Memory Space Requirement) การเข้าใจและนำหลักการเหล่านี้ไปใช้ในการจัดสรรไฟล์จะช่วยให้การใช้งานข้อมูลภายในฮาร์ดไดรฟ์เป็นไปอย่างมีประสิทธิภาพ เพิ่มความเสถียร ลดการสูญเสียเวลาในการค้นหาและเข้าถึงข้อมูล

1) การจัดสรรอย่างต่อเนื่อง (Contiguous Allocation) คือวิธีการที่มีประสิทธิภาพในการจัดเก็บข้อมูลไฟล์บนอุปกรณ์จัดเก็บข้อมูล หลักการนี้ประกอบด้วยการเรียงบล็อกหน่วยความจำที่ใช้สำหรับไฟล์เดียวกันให้ติดต่อกันโดยไม่มีช่องว่าง เพื่อให้การเข้าถึงข้อมูลสามารถทำได้เร็วยิ่งขึ้น หัวอ่านและเขียนของอุปกรณ์จัดเก็บข้อมูลจะไม่ต้องเคลื่อนที่ข้ามไปมาระหว่างบล็อกหน่วยความจำที่ห่างไกลกัน ทำให้เหมาะกับระบบไฟล์ที่มีโครงสร้างพื้นฐานและไม่ซับซ้อน อย่างไรก็ตาม การจัดสรรในลักษณะนี้อาจนำไปสู่ปัญหาเช่น การแตกหักของข้อมูล (fragmentation) และการใช้พื้นที่ไม่เต็มความสามารถ เมื่อมีการเปลี่ยนแปลงขนาดไฟล์อย่างต่อเนื่องดังภาพ 6 ซึ่งประกอบไปด้วยชุดข้อมูลจำลองที่มีแปดบล็อกจาก K1 ถึง K8 สะท้อนให้เห็นถึงการจัดสรรบล็อกหน่วยความจำที่ต่อเนื่องกันบนอุปกรณ์จัดเก็บข้อมูล



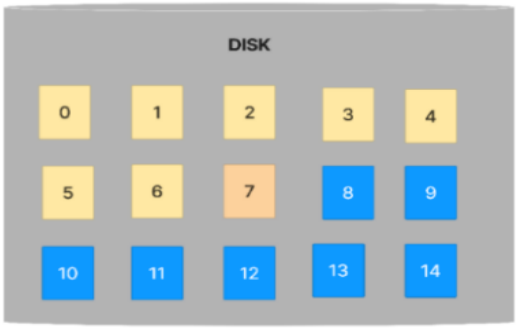
ภาพ 6 ชุดไฟล์ข้อมูลจำลองที่มีข้อมูลแปดช่วงตั้งแต่ K1 ถึง K8

หน่วยความจำรอง เช่น ฮาร์ดดิสก์ (Hard Disk) จะถูกจัดการโดยการแบ่งออกเป็นบล็อกดิสก์หรือเซกเตอร์ (sectors) ของข้อมูล เพื่อให้การเข้าถึงและจัดการข้อมูลเป็นไปอย่างมีระบบและมีประสิทธิภาพ



ภาพ 7 ภาพจำลองบล็อกดิสก์ของข้อมูล

การจัดสรรพื้นที่ดิสก์แบบต่อเนื่อง(Contiguous Allocation) เป็นเทคนิคของระบบปฏิบัติการใช้ในการจัดเก็บไฟล์ภายในหน่วยความจำดิสก์โดยการเรียงบล็อกข้อมูลอย่างต่อเนื่อง ซึ่งช่วยให้การเข้าถึงข้อมูลเป็นไปอย่างรวดเร็ว ในกรณีที่มีความจำเป็นต้องใช้พื้นที่จัดเก็บขนาด 8 บล็อก ระบบจะดำเนินการค้นหาบล็อกว่างที่ต่อเนื่องกันถึง 8 บล็อกและทำการจัดสรรให้กับไฟล์ที่ต้องการดังภาพ 8 การจัดสรรไฟล์ที่เริ่มต้นที่บล็อก 0 ต้องการพื้นที่ 8 บล็อก บล็อกดิสก์หมายเลข 0 ถึง 7 จึงถูกจัดสรรให้อย่างต่อเนื่อง และสามารถช่วยลดปัญหาการกระจัดกระจายของไฟล์และเพิ่มประสิทธิภาพในการเข้าถึงข้อมูล



ภาพ 8 ภาพการจัดสรรไฟล์อย่างต่อเนื่องภายในบล็อกดิสก์

ในระบบปฏิบัติการที่ใช้การจัดสรรพื้นที่ดิสก์แบบต่อเนื่อง (Contiguous Allocation) โครงสร้างของไดเร็กทอรี (Directory) จะบันทึกชื่อไฟล์ (File) ตำแหน่งเริ่มต้น (Start) และความยาวของไฟล์ (Length) ดังภาพ 9 ซึ่งประกอบด้วยรายการสำหรับไฟล์ Contiguous Allocation ที่มีตำแหน่งเริ่มต้นที่บล็อก 0 และมีความยาว 5 บล็อก การจัดสรรนี้ทำให้ระบบปฏิบัติการสามารถค้นหาและเข้าถึงไฟล์ได้อย่างรวดเร็วในอนาคต เนื่องจากมีข้อมูลเกี่ยวกับตำแหน่งและขนาดของไฟล์เก็บไว้ภายในไดเร็กทอรี ทันทีที่มีการบันทึกไฟล์ ระบบจะทำการค้นหาและจัดสรรบล็อกดิสก์ที่ต่อเนื่องกันเพื่อเก็บข้อมูล ซึ่งป้องกันการกระจัดกระจายของไฟล์ (File Fragmentation) และเพิ่มประสิทธิภาพในการเข้าถึงข้อมูลในระยะยาว

Directory

File	start	length
Contiguous Allocation	0	5

ภาพ 9 โครงสร้างไดเร็กทอรีของไฟล์ (Contiguous Allocation)

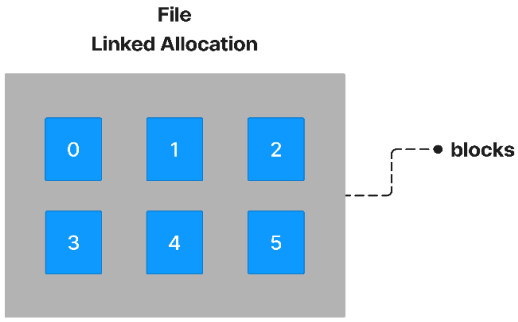
การจัดสรรพื้นที่แบบต่อเนื่องในระบบไฟล์มีไดเร็กทอรีบันทึกตำแหน่งเริ่มต้นและขนาดของไฟล์ ภาพที่ 10 แสดงไดเร็กทอรีที่บันทึกไฟล์ Contiguous Allocation เริ่มจากบล็อก 0 ครอบคลุม 8 บล็อก ไฟล์ B เริ่มต้นที่บล็อก 8 และครอบคลุมถึงบล็อก 12 มีความยาว 5 บล็อก และไฟล์ C เริ่มต้นที่บล็อก 16 และครอบคลุมถึงบล็อก 20 โดยมีความยาวเท่ากัน การจัดสรรดังกล่าวบนฮาร์ดดิสก์ สามารถช่วยเร่งความเร็วในการค้นหาและเข้าถึงข้อมูล ทั้งยังป้องกันการกระจัดกระจายของไฟล์เพื่อเพิ่มประสิทธิภาพการใช้งานระบบ

Directory

File	start	length
Contiguous Allocation	0	8
B	8	5
C	16	5

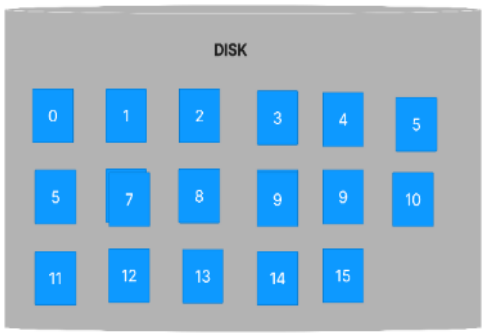
ภาพ 10 วิธีการจัดสรรที่ต่อเนื่องกันชุดไฟล์ส่วนเพิ่ม B และ C

2) การจัดสรรที่เชื่อมโยง (Linked Allocation) เป็นวิธีการจัดสรรพื้นที่ในระบบไฟล์ที่ใช้รายการเชื่อมโยงเพื่อเชื่อมต่อบล็อกข้อมูลที่ไม่ต้องติดต่อกันบนดิสก์ การจัดสรรนี้ช่วยลดปัญหาการแตกแฟรกเมนต์ภายนอกและเพิ่มความยืดหยุ่นในการจัดการพื้นที่จัดเก็บข้อมูลดังภาพ 11 สามารถเห็นได้ว่าไฟล์ถูกแบ่งออกเป็น 5 บล็อกที่มีการเข้าถึงตั้งแต่ 0 ถึง 4 และแต่ละบล็อกนั้นเชื่อมต่อกันผ่านการชี้ไปยังบล็อกถัดไปในลำดับ



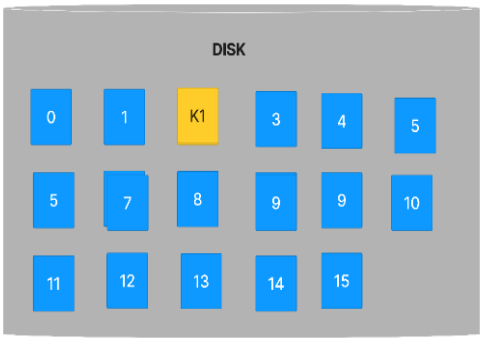
ภาพ 11 ข้อมูลภายในไฟล์ Linked Allocation จำนวน 5 บล็อก

หน่วยความจำรอง เช่น ฮาร์ดดิสก์ (Hard Disk) จะถูกจัดการโดยการแบ่งออกเป็นบล็อกดิสก์หรือเซกเตอร์ (sectors) ของข้อมูล เพื่อให้การเข้าถึงและจัดการข้อมูลเป็นไปอย่างมีระบบและมีประสิทธิภาพ



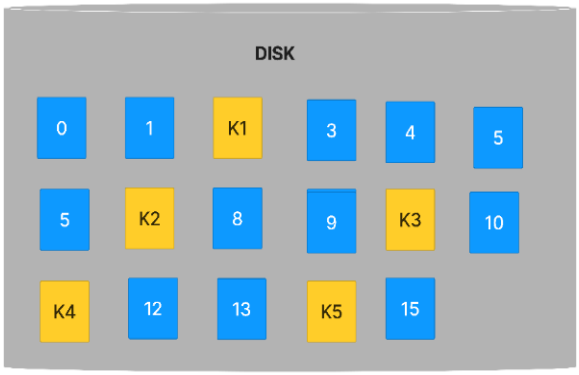
ภาพ 12 ภาพจำลองบล็อกดิสก์ของข้อมูล

ภาพ 13 บล็อก K1 ซึ่งตั้งอยู่บนบล็อกดิสก์หมายเลข 2 เป็นจุดเริ่มต้นของโครงสร้างการจัดเก็บแบบ Linked Allocation ที่บล็อกดิสก์ การจัดเก็บนี้ช่วยให้เกิดการเชื่อมโยงข้อมูลไฟล์ที่มีความยืดหยุ่นและเข้าถึงได้ต่อเนื่องโดยไม่ขึ้นกับตำแหน่งที่ต่อเนื่องกันของบล็อกดิสก์



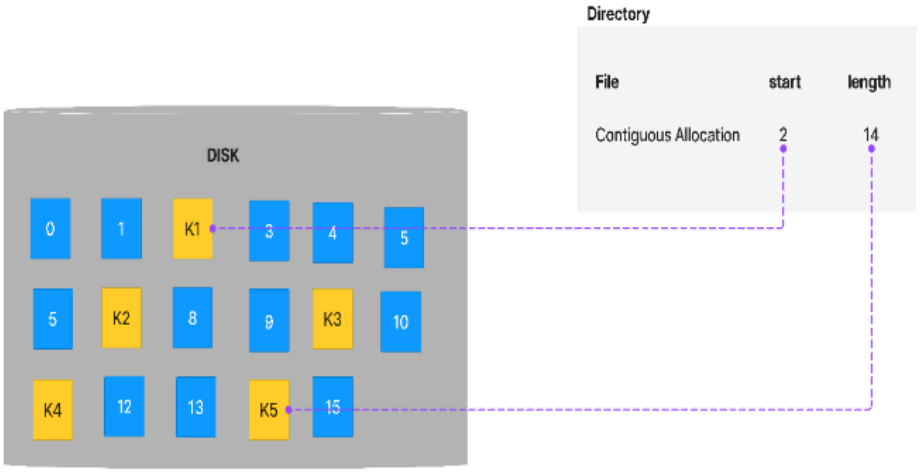
ภาพ 13 โครงสร้างบล็อกแรก K1 ใน Linked Allocation บนดิสก์บล็อก 2

การติดตามการจัดเก็บข้อมูลของไฟล์ดังภาพ 13 บล็อก K2 ได้รับการจัดเก็บที่ดิสก์บล็อก 7 ตามด้วยบล็อก K3 ที่ดิสก์บล็อก 9 บล็อก K4 ที่ดิสก์บล็อก 11 และสุดท้ายคือบล็อก B5 ที่ดิสก์บล็อก 14 การเรียงลำดับนี้สะท้อนถึงหลักการของการจัดเก็บแบบ Linked Allocation ที่ไม่จำเป็นต้องมีการเชื่อมต่อกันของบล็อกที่ติดกันในหน่วยความจำแต่เชื่อมโยงข้อมูลผ่านบล็อกดิสก์ที่ต่างกัน



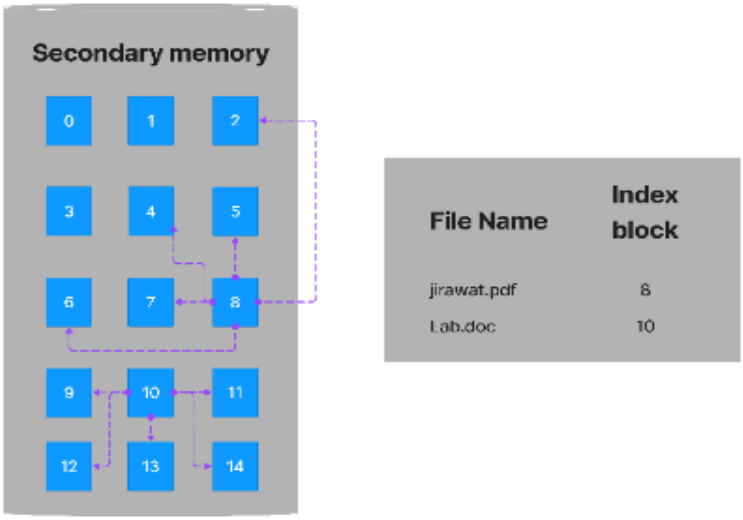
ภาพ 14 การจัดเก็บข้อมูลโครงสร้าง K2 จากดิสก์บล็อก 7 ถึงบล็อก B5 บนบล็อกดิสก์ 14

ในการจัดการดิสก์ภายใต้ระบบปฏิบัติการ (Operating System) การจัดสรรบล็อกดิสก์ (Disk Block Allocation) เพื่อการเก็บไฟล์ (File Storage) เป็นกระบวนการสำคัญที่ส่งผลต่อประสิทธิภาพในการค้นหาและเข้าถึงไฟล์ในภายหลัง ระบบปฏิบัติการมีหน้าที่รักษาไดเร็กทอรี (Directory) ที่มีรายการข้อมูลประกอบด้วยชื่อไฟล์ (File Name) บล็อกเริ่มต้น (Start Block) และความยาวของไฟล์ (File Length) ซึ่งมีการกำหนดตั้งแต่ลำดับที่ศูนย์ ในกรณีที่มีการเพิ่มไฟล์ระบบจะทำการค้นหาบล็อกที่ว่างและเรียงต่อเนื่องเพื่อการจัดสรรแบบต่อเนื่อง (Contiguous Allocation) ทำให้บล็อกของไฟล์ต่อเนื่องกัน ซึ่งช่วยให้การค้นหาและการเข้าถึงข้อมูลมีความรวดเร็วดังภาพ 15 โครงสร้างของไดเร็กทอรีและการจัดสรรบล็อกบนดิสก์



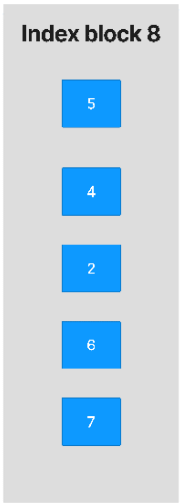
ภาพ 15 การจัดสรรบล็อกไฟล์และการเชื่อมโยงภายในบล็อกดิสก์

3) การจัดทำดรรชนี (Indexed Allocation) คือ กลวิธีการจัดการไฟล์คอมพิวเตอร์ที่เน้นการเพิ่มประสิทธิภาพในการเข้าถึงข้อมูล โดยใช้ตารางดรรชนีระบุตำแหน่งของบล็อกข้อมูลภายในหน่วยความจำรอง ซึ่งถูกจัดสรรเป็นหลายบล็อก ดังภาพที่ 16 หน่วยความจำรองถูกแบ่งเป็นบล็อกที่มีหมายเลข 0 ถึง 14 และตารางดรรชนีที่ระบุไฟล์ที่เกี่ยวข้องกับแต่ละบล็อก ตัวอย่างเช่น ไฟล์ jirawat.pdf อยู่ที่บล็อก 8 และไฟล์ Lab.doc อยู่ที่บล็อก 10 การใช้เทคนิคนี้ไม่เพียงแต่ช่วยลดการเสียพื้นที่จากการจัดเก็บไฟล์ในลำดับที่ไม่ต่อเนื่อง แต่ยังช่วยให้การค้นหาและเข้าถึงข้อมูลเป็นไปอย่างรวดเร็ว การจัดทำดรรชนีจึงเป็นกลยุทธ์สำคัญในการจัดการทรัพยากรหน่วยความจำของระบบไฟล์คอมพิวเตอร์อย่างมีประสิทธิภาพ



ภาพ 16 การจัดสรรไฟล์ที่จัดทำดรรชนี

ภาพ 17 บล็อกดรรชนีหมายเลข 8 ภายในระบบจัดการไฟล์ บล็อกนี้ไม่ได้จัดเก็บข้อมูลไฟล์โดยตรง แต่ใช้เป็นตัวบ่งชี้หรืออ้างอิงไปยังบล็อกที่จัดเก็บข้อมูลจริง การออกแบบดังกล่าวช่วยให้การค้นหาและการเข้าถึงไฟล์เป็นไปอย่างรวดเร็วและมีประสิทธิภาพ เนื่องจากระบบสามารถเข้าถึงบล็อกที่มีข้อมูลได้โดยตรงผ่านบล็อกดรรชนี กลวิธีนี้ลดความซับซ้อนของระบบไฟล์และลดการเรียกใช้ทรัพยากรของระบบคอมพิวเตอร์ ซึ่งท้ายที่สุดแล้วเพิ่มความเร็วและประสิทธิผลในการเข้าถึงข้อมูลไฟล์



ภาพ 17 โครงสร้างของดรรชนีบล็อก 8 และพอยน์เตอร์สำหรับเข้าถึงไฟล์ “jirawat.pdf”

การจัดสรรไฟล์ในฮาร์ดไดรฟ์อย่างมีประสิทธิภาพ (File Allocation in Hard Drives) จึงเป็นกระบวนการที่มีความสำคัญอย่างยิ่งต่อการบริหารจัดการพื้นที่จัดเก็บข้อมูลในฮาร์ดดิสก์ โดยมีเทคนิคหลักสำหรับการจัดสรรไฟล์ ได้แก่ การจัดสรรต่อเนื่อง (Contiguous Allocation) การจัดสรรเชื่อมโยง (Linked Allocation) และการทำดรรชนี (Indexing) แต่ละเทคนิคต่างมีข้อดีและข้อจำกัดที่แตกต่างกันทั้งในเรื่องของความยืดหยุ่น ความซับซ้อนในการบริหารจัดการ และประสิทธิภาพในการใช้งาน ดังนั้น การเลือกใช้เทคนิคที่เหมาะสมจึงมีความสำคัญอย่างยิ่งต่อการจัดการฮาร์ดไดรฟ์และไฟล์อย่างมีประสิทธิภาพ

2.5 ทฤษฎีการค้นหาเชิงเส้น

การค้นหาแบบเชิงเส้นหรือ Linear Search เป็นหนึ่งในอัลกอริทึมพื้นฐานที่สุดในด้านการค้นหาข้อมูล โดยมีหลักการทำงานที่เรียบง่าย คือ การสแกนหรือตรวจสอบข้อมูลแต่ละตัวอย่างในชุดข้อมูลตั้งแต่ต้นจนจบ เพื่อหาว่ามีข้อมูลตัวใดบ้างที่ตรงกับค่าที่กำหนดไว้ อัลกอริทึมนี้ไม่จำเป็นต้องใช้กับข้อมูลที่เรียงลำดับไว้ก่อนหน้า จึงสามารถนำไปประยุกต์ใช้ได้อย่างกว้างขวางในหลายสถานการณ์ (Knuth, 1998) ในการวิเคราะห์ประสิทธิภาพของอัลกอริทึมการค้นหา สิ่งสำคัญคือการพิจารณาถึงกรณีที่ดีที่สุด (Best Case) กรณีเฉลี่ย (Average Case) และกรณีที่ช้าที่สุด (Worst Case) สำหรับการค้นหาแบบเชิงเส้น กรณีที่ดีที่สุดคือเมื่อข้อมูลที่ต้องการค้นหาอยู่ที่ตำแหน่งแรกของชุดข้อมูล ในสถานการณ์นี้ ความซับซ้อนของเวลา (Time Complexity) จะเป็น $O(1)$ เนื่องจากอัลกอริทึมพบข้อมูลที่ต้องการทันทีหลังจากการตรวจสอบครั้งแรก (Cormen และคณะ, 2009) กรณีเฉลี่ย (Average Case) ของการค้นหาแบบเชิงเส้นเกิดขึ้นเมื่อข้อมูลที่ต้องการค้นหากระจายอยู่ในชุดข้อมูลอย่างสม่ำเสมอ ทำให้ต้องตรวจสอบข้อมูลโดยเฉลี่ยครึ่งหนึ่งของชุดข้อมูล ในกรณีนี้ ความซับซ้อนของเวลาจะเป็น $O(n)$ โดยที่ n คือจำนวนข้อมูลทั้งหมดในชุดข้อมูล (Sedgewick และWayne, 2011) แสดงให้เห็นว่าแม้กรณีเฉลี่ยจะมีประสิทธิภาพมากกว่ากรณีที่ช้าที่สุด แต่ก็ยังต้องพิจารณาถึงขนาดของชุดข้อมูลเนื่องจากการค้นหาแบบเชิงเส้นอาจไม่เหมาะกับชุดข้อมูลขนาดใหญ่ ในทางกลับกัน กรณีเลวร้ายที่สุดเกิดขึ้นเมื่อข้อมูลที่ต้องการค้นหาไม่มีอยู่ในชุดข้อมูล หรือ อยู่ที่ตำแหน่งสุดท้าย ทำให้อัลกอริทึมจำเป็นต้องตรวจสอบทุกตัวอย่างในชุดข้อมูล ส่งผลให้ความซับซ้อนของเวลาเพิ่มขึ้นเป็น $O(n)$ โดยที่ n คือ จำนวนข้อมูลทั้งหมดในชุดข้อมูล (Sedgewick และWayne, 2011) แม้ว่าการค้นหาแบบเชิงเส้น (Linear Search) จะมีประสิทธิภาพต่ำกว่าอัลกอริทึมการค้นหาที่ซับซ้อนกว่า เช่น การค้นหาแบบทวิภาค (Binary Search) ซึ่งต้องใช้ข้อมูลที่เรียงลำดับเรียบร้อยแล้ว หรืออัลกอริทึมที่ใช้โครงสร้างข้อมูลเฉพาะ เช่น เทบิลแฮช (Hash Tables) แต่การค้นหาแบบเชิงเส้นก็ยังมีความสำคัญในแง่ของความเรียบง่ายและความสามารถในการประยุกต์ใช้กับชุดข้อมูลทั่วไปที่ไม่จำเป็นต้องเรียงลำดับ (Knuth, 1998; Sedgewick และWayne, 2011)

2.6 ทฤษฎีการค้นหาแบบทวิภาค

ทฤษฎีการค้นหาแบบทวิภาคหรือ Binary Search Tree (BS) เป็นเทคนิคการค้นหาที่มีประสิทธิภาพสูงภายในอาร์เรย์ที่มีการเรียงลำดับข้อมูลแล้ว โดยกระบวนการนี้เริ่มต้นด้วยการกำหนดขอบเขตการค้นหาตั้งแต่ตำแหน่งแรก (low) ถึงตำแหน่งสุดท้าย (high) ของอาร์เรย์ จากนั้นจะคำนวณหาตำแหน่งกลาง (mid) ของขอบเขตการค้นหาและเปรียบเทียบค่าที่ต้องการค้นหา (k) กับค่าที่ตำแหน่งกลาง หากค่าที่ต้องการค้นหาตรงกับค่าที่ตำแหน่งกลาง การค้นหาจะสิ้นสุดทันทีด้วยผลลัพธ์ที่สำเร็จ แต่หากค่าที่ต้องการค้นหาน้อยกว่าค่าที่ตำแหน่งกลาง กระบวนการค้นหาจะต่อเนื่องในส่วนของอาร์เรย์ที่ตำแหน่งต่ำกว่ากลาง (จาก low ถึง mid-1) ในทางตรงกันข้าม หากค่าที่ต้องการค้นหามากกว่าค่าที่ตำแหน่งกลาง กระบวนการค้นหาจะต่อเนื่องในส่วนของอาร์เรย์ที่ตำแหน่งสูงกว่ากลาง (จาก mid+1 ถึง high) ทั้งนี้ กระบวนการค้นหาจะถูกทำซ้ำไปเรื่อยๆ จนกว่าจะพบค่าที่ต้องการหรือไม่มีข้อมูลเหลืออยู่ให้ค้นหา เนื่องจากในแต่ละขั้นตอนขอบเขตการค้นหาจะถูกลดลงครึ่งหนึ่ง ส่งผลทำให้เวลาที่ใช้ในการค้นหามีความเป็นไปได้ที่จะเป็นเชิงลอการิทึม (logarithm) ของจำนวนข้อมูลทั้งหมด $O(log\ n)$ ซึ่งแสดงให้เห็นถึงประสิทธิภาพสูงของการค้นหาแบบทวิภาคในการจัดการกับข้อมูลขนาดใหญ่ (Goodrich และTamassia, 2014)

2.7 ทฤษฎีต้นไม้ค้นหาแบบทวิภาค

ทฤษฎีต้นไม้แบบทวิภาคหรือ Binary Search Tree (BST) เป็นโครงสร้างข้อมูลที่มีความสำคัญในการจัดเก็บข้อมูลแบบมีการเรียงลำดับ ทุกโหนดภายในทฤษฎีต้นไม้แบบทวิภาคประกอบด้วยข้อมูลหรือค่า (value) และลิงก์ไปยังโหนดย่อยสองโหนด (node) คือโหนดย่อยทางซ้าย (left child) และโหนดย่อยทางขวา (right child) โดยที่ค่าในโหนดย่อยทางซ้ายจะต้องน้อยกว่าค่าในโหนดหลัก (parent node) ส่วนค่าในโหนดย่อยทางขวาจะต้องมากกว่าค่าในโหนดหลัก หลักการนี้ทำให้ทฤษฎีต้นไม้แบบทวิภาค สามารถดำเนินการค้นหา (search) การเพิ่ม (insert) และการลบ (delete) ข้อมูลได้อย่างมีประสิทธิภาพ โดยปกติการดำเนินการเหล่านี้มีเวลาดำเนินการเฉลี่ยอยู่ที่ $O(log\ n)$ โดยที่ n คือจำนวนโหนดทั้งหมดในต้นไม้ อย่างไรก็ตาม ประสิทธิภาพของทฤษฎีต้นไม้แบบทวิภาค อาจขึ้นอยู่กับความสมดุลของต้นไม้ หากต้นไม้มีความสมดุลไม่ดี การดำเนินการต่างๆ อาจมีประสิทธิภาพลดลงเป็นระดับเวลาที่ช้าที่สุดคือ $O(n)$ ซึ่งสามารถเกิดขึ้นได้ในกรณีที่ข้อมูลที่เพิ่มเข้ามามีลำดับที่เรียงกันอยู่แล้ว ทำให้ต้นไม้มีลักษณะคล้ายกับรายการเชื่อมโยง ดังนั้นการใช้งานทฤษฎีต้นไม้แบบทวิภาค จึงควรพิจารณาถึงลักษณะของข้อมูลที่จะจัดเก็บ เพื่อรักษาประสิทธิภาพการค้นหาให้เหมาะสม (Goodrich และTamassia, 2014)

2.8 ทฤษฎี Adelson–Velsky Landis Tree

ทฤษฎี Adelson–Velsky Landis Tree (AVL Tree) เป็นโครงสร้างข้อมูลแบบต้นไม้ค้นหาทวิภาค (Binary Search Tree, BST) ที่มีการปรับสมดุลตัวเอง (self–balancing) เพื่อให้ความลึกของต้นไม้ (depth of the tree) มีค่าใกล้เคียงกันที่สุด จึงทำให้การทำงานของการค้นหา

(search) การเพิ่ม (insert) และการลบ (delete) มีประสิทธิภาพใกล้เคียงกับ $O(log\ n)$ ในทุกกรณีโดยที่ n คือจำนวนโหนดภายในต้นไม้ สำหรับวิธีการค้นหา (Search) การค้นหาใน AVL Tree ทำงานเหมือนกับในต้นไม้แบบทวิภาคทั่วไป คือเริ่มต้นที่รากของต้นไม้ (Root node) แล้วเปรียบเทียบค่าที่ต้องการค้นหากับค่าในโหนดปัจจุบัน (Current node) หากค่าน้อยกว่า จะเคลื่อนไปยังโหนดลูกฝั่งซ้าย (Left child) หากมากกว่า จะเคลื่อนไปยังโหนดลูกฝั่งขวา (Right child) และหากเท่ากัน จะหยุดการค้นหา กระบวนการนี้จะทำซ้ำจนกว่าจะพบค่าหรือถึงโหนดที่ว่างเปล่า ซึ่งให้ความแน่นอนว่าประสิทธิภาพเวลาคือ $O(log\ n)$ ในกรณีเฉลี่ยและกรณีที่ช้าที่สุด เนื่องจากความสูงของต้นไม้ถูกจำกัดไว้ ส่วนของวิธีการเพิ่ม (Insert) การเพิ่มโหนดใหม่ลงใน AVL Tree จะเริ่มต้นด้วยการเพิ่มเหมือนกับในต้นไม้แบบทวิภาคธรรมดา โดยใส่โหนดใหม่เข้าไปในตำแหน่งที่เหมาะสม หลังจากนั้น ต้นไม้จะต้องถูกตรวจสอบและปรับสมดุลหากจำเป็น โดยการหมุน (rotations) ซึ่งมีสี่ประเภท คือการหมุนซ้าย (left rotation) การหมุนขวา (right rotation) การหมุนขวา-ซ้าย (right-left rotation) และการหมุนซ้าย-ขวา (left-right rotation) เพื่อให้คุณสมบัติของการสมดุล (balance property) ของ AVL Tree คงอยู่ นอกจากนี้วิธีการลบ (Delete) การลบโหนดจาก AVL Tree ค่อนข้างซับซ้อนกว่าการเพิ่ม เนื่องจากหลังลบโหนดออก อาจทำให้คุณสมบัติการสมดุลของต้นไม้เสียไป กระบวนการนี้ประกอบด้วยการค้นหาโหนดที่ต้องการลบ และอาจรวมถึงการใช้โหนดทดแทนจากลูกโหนดหรือโหนดที่อยู่ใกล้เคียงที่สุด หลังจากนั้น ต้องทำการปรับสมดุลต้นไม้เช่นเดียวกับในกระบวนการเพิ่ม ดังนั้น ทฤษฎี Adelson-Velsky Landis Tree (AVL Tree) เป็นการพัฒนาต่อจากโครงสร้างข้อมูลแบบต้นไม้ค้นหาทวิภาค ที่เน้นการรักษาความสมดุลของต้นไม้เพื่อให้การทำงานของการค้นหา เพิ่ม และลบข้อมูลมีประสิทธิภาพเป็น $O(log\ n)$ ในทุกกรณี อักทั้งสามารถปรับตัวได้อย่างรวดเร็วหลังจากการเพิ่มหรือลบโหนด จึงเป็นโครงสร้างข้อมูลที่เหมาะสมกับแอปพลิเคชันที่ต้องการการเข้าถึงข้อมูลอย่างรวดเร็วและมีประสิทธิภาพสูง (Goodrich และTamassia, 2014)

2.9 ทฤษฎีดรรชนีหลักสำหรับฐานข้อมูลเชิงสัมพันธ์

(Elmasri และNavathe, 2011) ดรรชนีหลักสำหรับฐานข้อมูลเชิงสัมพันธ์ (Primary Indexing for relational database) คือการสร้างดรรชนีให้กับคอลัมน์ (Column) ในตาราง (Table) เพื่อเพิ่มประสิทธิภาพในการทำงานของฐานข้อมูล ลดจำนวนการเข้าถึงฮาร์ดดิสก์เมื่อมีการประมวลผลคำสั่งแบบสอบถาม (Query) ทำให้สามารถค้นหาและเข้าถึงข้อมูลได้อย่างรวดเร็ว โดยแบ่งประเภทของการจัดทำดรรชนีดังนี้

2.9.1 การจัดทำดรรชนีระดับเดียว (Single Level Indexing)

2.10.1.1 การทำดรรชนีหลัก (Primary indexing) คือ ดรรชนีหลักในฐานข้อมูลเป็นตัวระบุเอกลักษณ์แถวประกอบด้วยคอลัมน์ที่มีค่าไม่ซ้ำและไม่ว่าง ซึ่งมีบทบาทสำคัญในการเพิ่มประสิทธิภาพการค้นหาและจัดการข้อมูลรวมถึงการเชื่อมต่อแต่ละตารางข้อมูล โดยดรรชนีหลักสามารถแบ่งออกได้เป็นสามประเภทหลักดังนี้

1) ดรรชนีแบบหนาแน่น (Dense Index) คือ โครงสร้างข้อมูลในระบบฐานข้อมูลที่จัดเก็บรายการเฉพาะทุกคีย์จากไฟล์ข้อมูลและชี้ไปยังบันทึกเต็มของข้อมูลนั้น ทำให้การค้นหาเป็นไปอย่างรวดเร็ว เนื่องจากมีการจัดเก็บข้อมูลครบถ้วนในแต่ละรายการ แต่อาจสิ้นเปลืองพื้นที่เก็บข้อมูลและต้องมีการปรับปรุงบ่อยครั้งเมื่อมีการเพิ่มหรือลบข้อมูล นอกจากนี้ยังช่วยให้การเข้าถึงข้อมูลมีประสิทธิภาพและเป็นระเบียบมากขึ้น โดยเฉพาะในระบบที่มีการจัดการข้อมูลขนาดใหญ่ 2) ดรรชนีแบบกระจาย (Sparse Index) คือ ดรรชนีแบบกระจายเป็นโครงสร้างข้อมูลที่เก็บรายการคีย์เฉพาะจากไฟล์ข้อมูลและชี้ไปยังบันทึกเต็มที่มีการเปลี่ยนแปลง ใช้พื้นที่เก็บข้อมูลน้อยกว่าดรรชนีแบบหนาแน่นเพราะไม่จำเป็นต้องบันทึกทุกรายการคีย์ ทำให้เหมาะสำหรับไฟล์ข้อมูลที่มีขนาดใหญ่หรือมีการเปลี่ยนแปลงบ่อยครั้ง แต่การค้นหาอาจช้ากว่าเนื่องจากต้องทำการค้นหาแบบลำดับในบางส่วนของข้อมูล ดรรชนีแบบกระจายช่วยลดภาระในการจัดการและอัปเดตข้อมูลโดยรักษาความสมดุลระหว่างความเร็วในการเข้าถึงข้อมูลและการใช้ทรัพยาก 3) การทำดรรชนีคลัสเตอร์ (Clustering Indexing) คือ เป็นวิธีการจัดเรียงบันทึกในฐานข้อมูลตามคีย์ดรรชนีที่เลือก ทำให้ข้อมูลที่มีค่าคีย์ใกล้เคียงกันจะถูกเก็บรวมกันในส่วนเดียวกันของพื้นที่จัดเก็บ ส่งผลให้การเข้าถึงข้อมูลที่มีคีย์ใกล้กันเป็นไปอย่างรวดเร็ว ลดเวลาในการค้นหาและเพิ่มประสิทธิภาพในการดึงข้อมูลหลายๆ รายการที่เกี่ยวข้องกัน แต่การปรับปรุงข้อมูลอาจต้องมีการเรียงลำดับใหม่ซึ่งอาจใช้เวลาและทรัพยากรมากขึ้น และ 4) ดรรชนีรอง (Secondary Indexing) คือ โครงสร้างข้อมูลที่ช่วยให้สามารถเข้าถึงข้อมูลในฐานข้อมูลได้เร็วขึ้นโดยการใช้คีย์ที่ไม่ใช่คีย์หลัก มันชี้ไปยังคีย์หลักหรือตำแหน่งของข้อมูล ช่วยในการค้นหาข้อมูลที่มีเงื่อนไขเฉพาะ แม้ว่าจะเพิ่มภาระในการบำรุงรักษา แต่ดรรชนีรองเพิ่มความสามารถในการสืบค้นและปรับปรุงการเข้าถึงข้อมูลที่ไม่ใช่ส่วนหลักของโครงสร้างข้อมูล

2.9.2 การจัดทำดรรชนีหลายระดับ (Multilevel Indexing) คือ กระบวนการสร้างดรรชนีภายในฐานข้อมูลที่มีการจัดระดับข้อมูลหลายชั้น เพื่อเพิ่มประสิทธิภาพในการค้นหาและเข้าถึงข้อมูล โดยทั่วไปจะประกอบด้วยดรรชนีหลัก (Primary Index) และดรรชนีรอง (Secondary Indexes) ดรรชนีหลักจะเชื่อมโยงโดยตรงกับข้อมูลในฐานข้อมูล ในขณะที่ดรรชนีรองจะช่วยเพิ่มความเร็วในการค้นหาข้อมูลที่มีคุณลักษณะเฉพาะ การใช้ดรรชนีหลายระดับช่วยลดเวลาที่จำเป็นสำหรับการค้นหาข้อมูลในฐานข้อมูลขนาดใหญ่ และทำให้การจัดการข้อมูลมีประสิทธิภาพมากขึ้น โดยเฉพาะในฐานข้อมูลที่มีข้อมูลจำนวนมหาศาลและต้องการการค้นหาที่ซับซ้อนหลากหลายมิติ

งานวิจัยที่เกี่ยวข้อง

1. Test–framework: Performance profiling and testing web search engine on non factoid queries. จากงานวิจัย (Althaf Ali และMahammad Shafi, 2019) เสนอเทคนิคใหม่ที่เรียกว่า Test Retrieval Framework สำหรับการทดสอบและวิเคราะห์ประสิทธิภาพของเครื่องมือค้นหาบนเว็บสำหรับคำถามที่ไม่ใช่แบบ factoid โดยใช้ข้อมูลที่เกี่ยวข้องที่ได้รับจากการค้นหาเพื่อทดสอบและทำนายความผิดพลาดและการทำงานปกติของเครื่องมือค้นหา

2. An Important Landmarks Construction for a GIS–Map based on Indexing of Dolly Images. จากงานวิจัย (Abdulkadhem และAl–Assadi, 2019) เสนอเทคนิคการสร้าง landmarks ที่สำคัญบนแผนที่ (Geographic Information System หรือ GIS) โดยอาศัยดรรชนีของภาพถ่ายดอลลี่ (Dolly Image) ผลการทดลองพบว่า เทคนิคที่เสนอนี้สามารถระบุตำแหน่งมุมระหว่างถนนสองสายขึ้นไปเป็น landmarks ได้อย่างมีประสิทธิภาพ

3. Efficient multi–keyword similarity search over encrypted cloud documents. จากงานวิจัย (Abdulsada, Honi และAl–Darraji, 2021) เสนอเทคนิคการค้นหาคำสำคัญหลายคำที่คล้ายกันบนข้อมูลที่เข้ารหัสในคลาวด์ ซึ่งสามารถให้ผลลัพธ์ที่ถูกต้องตามคะแนนความคล้ายคลึง และสนับสนุนการค้นหาข้อมูลอย่างปลอดภัยในคลาวด์

4. Hybridization of Pigeon inspired with glowworm'swarm optimization based clustering technique in wireless sensor networks. จากงานวิจัย (Alamelu และPrabu, 2022) เสนอเทคนิค Hybridization of Pigeon Inspired with Glowworm Swarm Optimization (HPIGSO) สำหรับการจัดกลุ่มในเครือข่ายเซ็นเซอร์ไร้สาย ซึ่งช่วยประหยัดพลังงานและเพิ่มอายุการใช้งานของเครือข่าย

5. Orthogonal frequency division multiplexing system with an indexed–pilot channel estimation. จากงานวิจัย (Alqatawneh, 2022) เสนอเทคนิค Orthogonal Frequency Division Multiplexing (OFDM) โดยใช้พายลอตช่องสัญญาณที่มีดรรชนี เพื่อส่งข้อมูลเพิ่มเติม โดยใช้ Minimum Mean Square Error (MMSE) ในการตรวจจับสัญลักษณ์พายลอต ผลการทดลองแสดงให้เห็นว่า สามารถใช้ดรรชนีของพายลอตเป็นศูนย์ เพื่อส่งข้อมูลเพิ่มเติมโดยไม่ส่งผลกระทบต่อความแม่นยำในการประมาณค่าแชนเนล

6. Clusterization of customer energy usage to detect power shrinkage in an effort to increase the efficiency of electric energy consumption. จากงานวิจัย (Asri และคณะ, 2021) เสนอเทคนิคการจัดกลุ่มข้อมูลการใช้ไฟฟ้าของลูกค้า เพื่อตรวจจับการสูญเสียไฟฟ้าที่ไม่ใช่ทางเทคนิค และเพิ่มประสิทธิภาพการใช้ไฟฟ้า ผลการทดลองแสดงให้เห็นว่าการจัดกลุ่มทั้งสี่กลุ่ม ให้ประสิทธิภาพดีที่สุดในการจัดกลุ่มข้อมูล ช่วยสนับสนุนการลดการสูญเสียทางไฟฟ้าในธุรกิจ

7. Dynamic optimization in binary search spaces via weighted superposition attraction algorithm. จากงานวิจัย (Baykasoğlu และOzsoydan, 2018) เสนอเทคนิค Weighted Superposition Attraction Algorithm (WSA) สำหรับแก้ปัญหาการเพิ่มประสิทธิภาพแบบพลวัตในพื้นที่ค้นหาแบบทวิภาค โดย WSA เป็นอัลกอริทึมเชิงวิวัฒนาการใหม่ที่มีประสิทธิภาพสูง ช่วยสนับสนุนงานด้านการเพิ่มประสิทธิภาพแบบพลวัตในระบบผลิตที่ยืดหยุ่น

8. Formulation of city health development index using data mining. จากงานวิจัย (Bertalya และคณะ, 2021) เสนอเทคนิคการพัฒนาดรรชนีสุขภาพเมืองโดยใช้เทคนิคการทำเหมืองข้อมูล ซึ่งช่วยสนับสนุนการวางแผนและการตัดสินใจด้านสาธารณสุข

9 . An Ultra-Scalable Blockchain Platform for Universal Asset Tokenization: Design and Implementation. จากงานวิจัย (Buldas และคณะ, 2022) เสนอเทคนิคบล็อกเชน (Blockchain) ที่มีขนาดใหญ่ ไม่จำกัด และรองรับการใช้งานหลากหลาย เพื่อสนับสนุนกระบวนการแปลงสินทรัพย์หรือข้อมูลต่างๆ (tokenization) อย่างไร้ขีดจำกัด

10. Optimization of Index-Based Method of Metadata Search for Large-Scale File Systems. จากงานวิจัย (Chang และคณะ, 2017) เสนอเทคนิคการค้นหาข้อมูลอภิพันธุ์ (Metadata) บนระบบไฟล์ขนาดใหญ่ โดยใช้การจัดหมวดหมู่ดรรชนี (index) แบบลำดับชั้น (hierarchical) เพื่อเพิ่มประสิทธิภาพการค้นหา ผลการทดลองแสดงให้เห็นว่าเทคนิคนี้ช่วยเพิ่มประสิทธิภาพการค้นหา metadata ได้ดีกว่าเทคนิคเดิม

11. MongoDB Indexing for Performance Improvement. จากงานวิจัย (Chopade และ Pachghare, 2020) เสนอเทคนิคการสร้างดรรชนี (index) ในฐานข้อมูลที่เรียกว่า MongoDB เพื่อเพิ่มประสิทธิภาพการค้นหาข้อมูล ผลการทดลองแสดงให้เห็นว่าการสร้าง index สามารถลดเวลาในการค้นหาข้อมูลลงได้อย่างมาก เทคนิคนี้จึงช่วยปรับปรุงประสิทธิภาพการทำงานของฐานข้อมูล (database) ได้เป็นอย่างดี

12. New Path Based Index Structure for Processing CAS Queries over XML Database. จากงานวิจัย (Gopinathan และAsawa, 2017) เสนอเทคนิคการสร้างโครงสร้างดรรชนีใหม่ที่รวมพี่น้องของโหนดปลายเป็นเส้นทางเดียว เพื่อประมวลผลคำสั่งค้นหาแบบต้นไม้ (Twig queries) ได้อย่างมีประสิทธิภาพมากขึ้น ช่วยสนับสนุนการประมวลผลคำสั่งค้นหาข้อมูลแบบโครงสร้างและเนื้อหาบนฐานข้อมูล (Extensible Markup Language , XML)

13. The next 50 years in database indexing or: the case for automatically generated index structures. จากวิจัย (Dittrich, Nix และSchön, 2021) เสนอเทคนิคการสร้างดรรชนีฐานข้อมูลโดยใช้ขั้นตอนวิธีพันธุกรรม ซึ่งช่วยปรับปรุงประสิทธิภาพการค้นหาและประมวลผลข้อมูล

14. Linear-time String Indexing and Analysis in Small Space. จากงานวิจัย (Belazzougui และคณะ, 2020) เสนอเทคนิคการสร้างดรรชนี (Index) และวิเคราะห์สตริงในเวลาเชิงเส้น โดยใช้พื้นที่จำกัด เพื่อสนับสนุนการค้นหาและวิเคราะห์ข้อมูลขนาดใหญ่อย่างรวดเร็วและประหยัดพื้นที่

15. Large-Scale Indexing, Discovery, and Ranking for the Internet of Things (IoT). จากงานวิจัย (Fathy, Barnaghi และTafazolli, 2018) เสนอเทคนิคในการจัดทำดรรชนีค้นหา และจัดอันดับข้อมูล (Internet of Things, IoT) ขนาดใหญ่ ซึ่งช่วยสนับสนุนการพัฒนาและประยุกต์ใช้งาน IoT ในรูปแบบที่ดรรชนีแบบกระจายและขนาดใหญ่

16. The PGM-index: a multicriteria, compressed and learned approach to data indexing จากงานวิจัย (Ferragina และVinciguerra, 2019) เสนอเทคนิคดรรชนีที่เรียกว่า (Piecewise Geometric Model index, PGM-index) ซึ่งเป็นโครงสร้างข้อมูลที่ช่วยในการค้นหาและปรับปรุงข้อมูลอย่างรวดเร็ว โดยใช้พื้นที่น้อยกว่าดรรชนีแบบดั้งเดิม และยังมีการรับประกันเวลาค้นหาในกรณีที่กรณีที่ช้าที่สุด ซึ่งช่วยสนับสนุนงานด้านการจัดเก็บและค้นหาข้อมูลอย่างมีประสิทธิภาพ

17. A Practical Framework for Secure Document Retrieval in Encrypted Cloud File Systems. จากงานวิจัย (Fu และคณะ, 2021) เสนอเทคนิคกรอบการทำงานเพื่อการเรียกค้นเอกสารอย่างปลอดภัยในระบบไฟล์คลาวด์ที่เข้ารหัส ผลการวิจัยช่วยสนับสนุนการเก็บข้อมูลอย่างปลอดภัยในคลาวด์

18. Big Data retrieval techniques based on Hash Indexing and MapReduce approach with NoSQL Database. จากงานวิจัย (Gayathiri, Jaspher และNatarajan, 2019) เสนอเทคนิค Hash Indexing และ MapReduce เพื่อการค้นคืนข้อมูล Big Data อย่างรวดเร็ว โดยใช้ฐานข้อมูลไม่ใช่เชิงสัมพันธ์ (Not only Structured Query Language, NoSQL) ซึ่งช่วยสนับสนุนการจัดการและวิเคราะห์ข้อมูลขนาดใหญ่

19. Information extraction model from Ge'ez texts. จากงานวิจัย (Gebeyehu, Wolde และS. Shibeshi, 2023) เสนอเทคนิคการสกัดข้อมูลจากข้อความภาษา Ge'ez โดยใช้อัลกอริทึมการเรียนรู้ของเครื่อง เพื่อช่วยสนับสนุนงานวิจัยด้านภาษาศาสตร์และประวัติศาสตร์

20. Penguins Search Optimisation Algorithm for Association Rules Mining. จากงานวิจัย (Gheraibia และคณะ, 2016) เสนอเทคนิคอัลกอริทึมใหม่สำหรับการทำเหมืองกฎความสัมพันธ์ โดยใช้ penguins search optimization algorithm เพื่อเพิ่มประสิทธิภาพและลดเวลาในการค้นหากฎความสัมพันธ์จากฐานข้อมูล ซึ่งช่วยให้ได้กฎความสัมพันธ์ที่มีคุณภาพสูงและมีความสอดคล้องกันมากขึ้น

21. Efficient Implementation for Deterministic Finite Tree Automata Minimization. จากงานวิจัย **(Guellouma และCherroun, 2016)** เสนอเทคนิคใหม่ในการปรับปรุงการลดขนาดของออโตมาตันต้นไม้จำกัดแบบกำหนด (Deterministic Finite Tree Automata) โดยใช้กราฟที่นิยามได้ชัดเจน เพื่อเพิ่มประสิทธิภาพในการประมวลผล ซึ่งช่วยสนับสนุนงานด้านการประมวลผลภาษาธรรมชาติและการวิเคราะห์โครงสร้างข้อมูล

22. Exploring Index Structures for Zoned Namespaces SSDs. จากงานวิจัย (Jin และคณะ, 2021) เสนอเทคนิคการสร้างโครงสร้างดรรชนีสำหรับ Zoned Namespaces Solid State Drives เพื่อเพิ่มประสิทธิภาพการอ่านเขียนข้อมูล ซึ่งช่วยลดเวลาการอ่านเขียนข้อมูลได้อย่างมีนัยสำคัญ รวมถึงสนับสนุนการใช้งาน Zoned Namespaces Solid State Drives อย่างมีประสิทธิภาพมากยิ่งขึ้น

23. Data mining, fuzzy AHP and TOPSIS for optimizing taxpayer supervision. จากงานวิจัย (Jin และคณะ, 2021) เสนอเทคนิคการทำเหมืองข้อมูล การวิเคราะห์เชิงลำดับชั้นคลุมเครือ และ Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS) เพื่อเพิ่มประสิทธิภาพการตรวจสอบผู้เสียภาษี

24. An approach for enhancing data confidentiality in Hadoop. จากงานวิจัย (Kareem, Yousif และJihad Abdalwahid, 2020) เสนอเทคนิคการเข้ารหัสข้อมูลแบบผสมระหว่าง Rivest Shamir Adleman และ Paillier Pascal Paillier เพื่อเพิ่มความปลอดภัยของข้อมูลใน Hadoop ซึ่งช่วยเพิ่มความมั่นคงปลอดภัยของระบบชุดข้อมูลขนาดใหญ่

25. Random Binary Search Trees for approximate nearest neighbour search in binary spaces. จากงานวิจัย (Komorowski และTrzciński, 2019) เสนอเทคนิคการใช้ Random Binary Search Trees สำหรับการค้นหาเพื่อนบ้านที่ใกล้ที่สุดโดยประมาณในพื้นที่ข้อมูลทวิภาค ซึ่งช่วยสนับสนุนงานด้านการค้นหาข้อมูลที่มีมิติสูง

26. The Case for Learned Index Structures. จากงานวิจัย (Kraska และคณะ, 2018) เสนอเทคนิคการใช้โมเดลการเรียนรู้แทนโครงสร้างดรรชนีแบบดั้งเดิม เพื่อปรับปรุงประสิทธิภาพการค้นหาข้อมูล ผลพบว่าโมเดลการเรียนรู้สามารถทำงานได้เร็วขึ้นและใช้พื้นที่น้อยกว่าโครงสร้างดรรชนีแบบดั้งเดิม

27. Space-Efficient Index Scheme for PCM-Based Multiversion Databases in Cyber-Physical Systems. จากงานวิจัย (Kuan และคณะ, 2016) เสนอเทคนิคการสร้างรูปแบบดรรชนี (Index scheme) สำหรับฐานข้อมูล Multiversion บน (Phase-change memory, PCM) เพื่อใช้งานกับ (Cyber-physical systems, CPSs) โดยเสนอ (Space-efficient multiversion index, SEMI) เพื่อเพิ่มประสิทธิภาพการใช้พื้นที่และประสิทธิภาพการเข้าถึงข้อมูล ซึ่งช่วยสนับสนุนการทำงานของระบบฐานข้อมูลแบบทันทีใน CPSs

28. Towards Data-Based Cache Optimization of B+-Trees. จากงานวิจัย (Kühn และคณะ, 2023) เสนอเทคนิคการเพิ่มประสิทธิภาพหน่วยความจำแบบ Cache สำหรับ (Balanced Plus Trees, B+-Trees) โดยใช้ข้อมูลเชิงสถิติ ผลการทดลองแสดงให้เห็นว่าเทคนิคดังกล่าวสามารถเพิ่มประสิทธิภาพการเข้าถึงหน่วยความจำ Cache ได้ดีขึ้น ซึ่งช่วยสนับสนุนงานด้านการออกแบบโครงสร้างข้อมูลฐานข้อมูลและการเข้าถึงหน่วยความจำอย่างมีประสิทธิภาพ

29. Fast surveillance video indexing & retrieval with WiFi MAC address tagging. จากงานวิจัย (Tan และLim, 2019) เสนอเทคนิคการใช้ (Wireless Packet Sniffer, WiFi Sniffer) และการติดตามที่อยู่ของอุปกรณ์ (Media Access Control address, MAC address) เพื่อดรรชนีและค้นหาวิดีโอเฝ้าระวังอย่างรวดเร็ว ช่วยสนับสนุนงานด้านความปลอดภัยและการเฝ้าระวัง

30. A streaming clustering approach using a heterogeneous system for big data analysis. จากงานวิจัย (Tan และLim, 2019) เสนอเทคนิคการวิเคราะห์ข้อมูลขนาดใหญ่ด้วยการจัดกลุ่มแบบสตรีมมิ่งใช้ระบบเฮตีโรเจนีอัส (heterogeneous) ซึ่งช่วยในการวิเคราะห์ข้อมูลขนาดใหญ่อย่างมีประสิทธิภาพ

31. A novel method for parallel indexing of real time geospatial big data generated by IoT devices. จากงานวิจัย (Lee และคณะ, 2017) เสนอเทคนิคการสร้าง (Rectangle Tree, R-Tree) แบบขนานโดยใช้ Apache Spark เพื่อดรรชนีข้อมูลภูมิศาสตร์แบบเรียลไทม์ (Real-time) จากอุปกรณ์ที่เรียกว่า (Internet of Things, IoT) ซึ่งช่วยจัดการข้อมูลขนาดใหญ่ได้อย่างมีประสิทธิภาพ

32. Fast Online Similarity Search for Uncertain Time Series. จากงานวิจัย (Ma, Zheng และYan, 2020) เสนอเทคนิคการค้นหาความคล้ายคลึงออนไลน์สำหรับชุดข้อมูลเวลาที่ไม่แน่นอน เพื่อเพิ่มประสิทธิภาพการค้นคืนข้อมูลออนไลน์ ผลการทดลองแสดงให้เห็นว่าโครงสร้างดรรชนีสามารถปรับปรุงได้อย่างรวดเร็วในสภาพแวดล้อมเวลาจริง และช่วยสนับสนุนงานด้านการประมวลผลข้อมูลเวลาออนไลน์

33. A Hybrid Approach for Clustering Uncertain Time Series. จากงานวิจัย (Ma, Zhu และYan, 2020) เสนอเทคนิคการจัดกลุ่มแบบลำดับชั้นโดยแบ่งข้อมูลออกเป็นกลุ่มย่อยๆ ก่อนแล้วจึงรวมกลุ่มย่อยเหล่านั้น ซึ่งช่วยเพิ่มประสิทธิภาพและความแม่นยำในการจัดกลุ่มข้อมูลเวลาที่ไม่แน่นอนเมื่อเทียบกับเทคนิค UK-means แบบดั้งเดิม

34. In-Memory Distributed Indexing for Large-Scale Media Data Retrieval. จากงานวิจัย (Ma และคณะ, 2017) เสนอเทคนิคดรรชนีสร้างขึ้นจากการดรรชนีแบบกระจายความจำเพื่อการค้นหารูปภาพมีประสิทธิภาพ ซึ่งช่วยสนับสนุนการค้นหารูปภาพขนาดใหญ่และเร็วขึ้น

35. Analyses of Indexing Techniques on Uncertain Data With High Dimensionality. จากงานวิจัย (Ibrahim, Sani และYaakob, 2020) เสนอเทคนิคการวิเคราะห์เปรียบเทียบประสิทธิภาพของเทคนิคการสร้างดรรชนีสำหรับข้อมูลที่มีความไม่แน่นอนและมิติสูง เพื่อหาโครงสร้างการจัดทำดรรชนีที่เหมาะสมที่สุด ซึ่งช่วยเพิ่มประสิทธิภาพการจัดเก็บและค้นหาข้อมูล

36. A fuzzy neighborhood rough set method for anomaly detection in large scale data. จากงานวิจัย (Elhoussaine, 2020) เสนอเทคนิคการตรวจจับค่าผิดปกติในข้อมูลขนาดใหญ่โดยใช้ทฤษฎีชุดข้อมูลหยาบแบบฟัซซี่ (fuzzy neighborhood rough set) ซึ่งช่วยในการวิเคราะห์ข้อมูลขนาดใหญ่และตรวจจับค่าผิดปกติอย่างมีประสิทธิภาพ ซึ่งช่วยสนับสนุนงานด้านการทำเหมืองข้อมูลและการเรียนรู้เชิงลึก

37. C4.5 Decision Tree Algorithm for Spatial Data, Alternatives and Performances. จากงานวิจัย (Oujdi, Belbachir และBoufares, 2019) เสนอเทคนิคการปรับใช้อัลกอริทึมต้นไม้ตัดสินใจ C4.5 ให้เหมาะสมกับข้อมูลทางภูมิศาสตร์ เรียกว่า S-C4.5 ซึ่งช่วยสนับสนุนการจัดการและวิเคราะห์ข้อมูลทางภูมิศาสตร์ได้ดีขึ้น

38. Performance evaluation of cloud service with hadoop for twitter data. จากงานวิจัย (Panatula และคณะ, 2019) เสนอเทคนิคการประเมินประสิทธิภาพของ Hadoop Map Reduce ในการประมวลผลข้อมูล Twitter พบว่า Hadoop มีประสิทธิภาพสำหรับแอปพลิเคชัน Big Data ที่มีมากกว่าสามโหนดขึ้นไปของปัจจัยการจำลองทั้งสามรูปแบบ ซึ่งช่วยสนับสนุนงานด้านการจัดการและวิเคราะห์ข้อมูลขนาดใหญ่

39. Quality and energy optimized scheduling technique for executing scientific workload in cloud computing environment. จากงานวิจัย (Prasad และKulkarni, 2021) เสนอเทคนิคการจัดตารางงานที่เน้นประสิทธิภาพด้านคุณภาพและพลังงานสำหรับการประมวลผลงานทางวิทยาศาสตร์ในสภาพแวดล้อมคลาวด์ ซึ่งสามารถเพิ่มประสิทธิภาพการใช้ทรัพยากรและลดการใช้พลังงานได้ สนับสนุนงานวิจัยด้านการจัดการทรัพยากรและการจัดสรรงานในระบบคลาวด์

40. A Comparative Study of Secondary Indexing Techniques in LSM-based NoSQL Databases. จากงานวิจัย (Qader, Cheng และHristidis, 2018) เสนอการศึกษาเปรียบเทียบเทคนิคการสร้างดรรชนีรองในฐานข้อมูลที่ไม่ใช่เชิงสัมพันธ์ (Not only Structured Query Language, JavaScript Object Notation, NoSQL) ที่มีพื้นฐานเป็น (Log-Structured Merge-tree, LSM) โดยให้คำแนะนำทางปฏิบัติในการเลือกดรรชนีรอง และสนับสนุนการพัฒนาและปรับปรุงประสิทธิภาพของฐานข้อมูล NoSQL ที่ใช้ LSM เป็นพื้นฐาน

41. Wind-Bell Index: Towards Ultra-Fast Edge Query for Graph Databases. จากงานวิจัย (Qiu และคณะ, 2023) เสนอเทคนิคที่เรียกว่า Wind-Bell Index สำหรับการค้นหาขอบเขตอย่างรวดเร็วในฐานข้อมูลกราฟ ซึ่งช่วยสนับสนุนการดำเนินงานทางด้านการจัดการข้อมูลกราฟมีประสิทธิภาพมากขึ้น

42. Cluster head selection based on Minimum Connected Dominating Set and Bi-Partite inspired methodology for energy conservation in WSNs. จากงานวิจัย (Priyadarshini และSivakumar, 2021) เสนอเทคนิคการเลือกหัวหน้ากลุ่มจาก Minimum Connected Dominating Set และ Bi-Partite เพื่อประหยัดพลังงานใน (Wireless Sensor Networks, WSNs) ซึ่งช่วยเพิ่มอายุการใช้งานเครือข่ายและลดความล่าช้า

43. Indexing intelligence using benchmark classifier. จากงานวิจัย (S และMb, 2020) เสนอเทคนิคการใช้โมเดลการจัดประเภท (k-Nearest Neighbors, k-NN) เพื่อวิเคราะห์สัญญาณ (Electroencephalogram, EEG) ในการวัดความคล้ายคลึงกับข้อมูลมาตรฐาน ซึ่งสามารถช่วยในการวิเคราะห์ความฉลาดของมนุษย์และสนับสนุนงานด้านวิทยาศาสตร์การแพทย์

44. B+-Tree Based Multi-Keyword Ranked Similarity Search Scheme Over Encrypted Cloud Data. จากงานวิจัย (Shen และคณะ, 2021) เสนอเทคนิคการค้นหาคำหลายคำที่มีการจัดอันดับตามความคล้ายคลึงบนข้อมูลที่ถูกเข้ารหัสในคลาวด์ โดยใช้โครงสร้างข้อมูล (Balanced Plus Trees, B+-Trees) เพื่อสนับสนุนความเร็วในการค้นหาและความปลอดภัยของข้อมูล

45. Join optimization for inverted index technique on relational database management systems. จากงานวิจัย (Shin, Ahn และIm, 2022) เสนอเทคนิคที่เรียกว่า Join Optimization for Inverted Index บนระบบการจัดการฐานข้อมูลเชิงความสัมพันธ์ เพื่อปรับปรุงประสิทธิภาพการค้นหาคำหลัก ซึ่งช่วยสนับสนุนงานด้านการค้นหาและการจัดการข้อมูลอย่างมีประสิทธิภาพ

46. A text mining and topic modeling based bibliometric exploration of information science research. จากงานวิจัย (Silwattananusarn และ Kulkanjanapiban, 2022) เสนอเทคนิคการวิเคราะห์บรรณานุกรมและการทำเหมืองข้อมูลข้อความ โดยใช้เรื่องราวของวิทยาศาสตร์สารสนเทศ เพื่อสำรวจแนวโน้มและพัฒนาการของงานวิจัยด้านนี้ ซึ่งช่วยในการค้นหาแนวโน้มและเส้นทางใหม่ในงานวิจัยด้านสารสนเทศศาสตร์

47. A Distance-Computation-Free Search Scheme for Binary Code Databases. จากงานวิจัย (Song และคณะ, 2016) เสนอเทคนิคการค้นหาแบบไม่ต้องคำนวณระยะห่างสำหรับฐานข้อมูลโค้ดไบนารี ผลการทดลองแสดงให้เห็นว่าเทคนิคนี้มีประสิทธิภาพในการค้นหาสูง และช่วยสนับสนุนงานด้านการค้นหาแบบใกล้เคียง

48. A novel feature engineering algorithm for air quality datasets. จากงานวิจัย (Hashem และPillai, 2020) เสนอเทคนิคอัลกอริทึมสำหรับวิศวกรรมคุณลักษณะสำหรับชุดข้อมูลคุณภาพอากาศ เพื่อช่วยในการวิเคราะห์และพยากรณ์มลพิษทางอากาศ

49. Dumpy: A Compact and Adaptive Index for Large Data Series Collections. จากงานวิจัย (Wang และคณะ, 2023) เสนอเทคนิคที่เรียกว่า Dumpy ซึ่งใช้โครงสร้างข้อมูลหลายทางแบบปรับตัวได้และอัลกอริทึมแบ่งโหนดแบบปรับตัวได้ เพื่อสร้างดรรชนีที่มีประสิทธิภาพ ผลการทดลองแสดงให้เห็นว่า Dumpy สามารถลดเวลาในการสร้างดรรชนีและเพิ่มความแม่นยำในการค้นหาเมื่อเทียบกับวิธีการอื่นๆ งานวิจัยนี้จะช่วยสนับสนุนการจัดการและวิเคราะห์ข้อมูลอนุกรมเวลาขนาดใหญ่อย่างมีประสิทธิภาพ

50. Twitter data analysis using hadoop ecosystems and apache zeppelin. จากงานวิจัย (Stanly Wilson, 2019) เสนอเทคนิคการวิเคราะห์ข้อมูลจาก Twitter โดยใช้ Hadoop ecosystem และ Apache Zeppelin เพื่อช่วยสนับสนุนการวิเคราะห์ข้อมูลขนาดใหญ่

51. A Performance-Improved and Storage-Efficient Secondary Index for Big Data Processing. จากงานวิจัย (Wu และคณะ, 2017) เสนอเทคนิคดรรชนีรองสำหรับข้อมูลขนาดใหญ่ เพื่อปรับปรุงประสิทธิภาพการสืบค้นและการเข้าถึงข้อมูล ช่วยสนับสนุนงานวิเคราะห์ข้อมูลขนาดใหญ่

52. A novel index structure for large scale image descriptor search. จากงานวิจัย (Yuan และLiu, 2012) เสนอเทคนิค โครงสร้างดรรชนีใหม่สำหรับการค้นหารูปภาพขนาดใหญ่ โดยใช้ Embedded k–Means ซึ่งเป็นวิธีการค้นหาเพื่อนบ้านที่ใกล้ที่สุดโดยประมาณ ซึ่งช่วยสนับสนุนงานทางด้านการค้นหารูปภาพและการจัดเก็บข้อมูลรูปภาพขนาดใหญ่

53. Efficiency of JSON for Data Retrieval in Big Data. จากงานวิจัย (Yusof และMan, 2017) เสนอเทคนิคการใช้ (JavaScript Object Notation, JSON) ในการสืบค้นข้อมูลขนาดใหญ่ พบว่า JSON มีประสิทธิภาพดีกว่า (eXtensible Markup Language, XML) ในการสืบค้นข้อมูลเร็วขึ้นและใช้ทรัพยากรหน่วยประมวลผลกลาง (Central Processing Unit, CPU) น้อยกว่า ซึ่งช่วยสนับสนุนการจัดการและวิเคราะห์ข้อมูลขนาดใหญ่

54. Optimizing random forest classifier with Jenesis–index on an imbalanced dataset. จากงานวิจัย (Zeffora และShobarani, 2022) เสนอเทคนิคการเพิ่มประสิทธิภาพตัวจำแนก Random forest โดยใช้ดรรชนี Jenesis บนชุดข้อมูลที่ไม่สมดุล พบว่าผลลัพธ์ดีขึ้นในการจำแนกข้อมูล และช่วยสนับสนุนงานด้านการเรียนรู้ของเครื่อง

55. Secure and Efficient Product Information Retrieval in Cloud Computing. จากงานวิจัย (Zhao และZeng, 2018) เสนอเทคนิคการค้นหาข้อมูลผลิตภัณฑ์ที่ปลอดภัยและมีประสิทธิภาพในการคำนวณคลาวด์ โดยผลการวิเคราะห์และจำลองแสดงความปลอดภัยและประสิทธิภาพของระบบนี้ ซึ่งสามารถสนับสนุนงานด้านความปลอดภัยของข้อมูลและการค้นหาข้อมูลผลิตภัณฑ์ในสภาพแวดล้อมคลาวด์คอมพิวติ้ง

56. Indexing Multimedia Data with an Extension of Binary Tree –– Image Search by Content ––. จากงานวิจัย (Zineddine, Amine และAdeel, 2021) เสนอเทคนิคใหม่ในการสร้างดรรชนีข้อมูลมัลติมีเดีย โดยใช้การขยายของ Binary tree เพื่อช่วยในการค้นหาภาพโดยอิงเนื้อหา การใช้เทคนิคนี้ทำให้การค้นหาข้อมูลมัลติมีเดียมีประสิทธิภาพยิ่งขึ้น และสามารถค้นหาภาพที่มีเนื้อหาที่คล้ายคลึงกันได้ด้วยความรวดเร็วและความแม่นยำที่สูง

นอกจากนี้ยังมีงานวิจัยที่เกี่ยวข้องกับการจัดทำดรรชนีบนเทคโนโลยี Hadoop Distributed File System (HDFS) ซึ่งเป็นระบบเครือข่ายคอมพิวเตอร์ที่กระจายหลายเครื่องได้รับการออกแบบมาเพื่อจัดเก็บและประมวลผลข้อมูลขนาดใหญ่ เช่น (Chen และคณะ, 2021) เสนอเทคนิค Small File Merger (SFM) เพื่อแก้ปัญหาการเข้าถึงไฟล์ขนาดเล็กอย่างมีประสิทธิภาพบน HDFS โดยการรวมไฟล์ขนาดเล็กเข้าด้วยกันและสร้างดัชนีสำหรับการเข้าถึงไฟล์ (Wu และคณะ, 2022) เสนอเทคนิค A multi-indexing system based on HDFS for remote sensing data storage (MIX-RS) มุ่งเน้นการใช้ระบบดรรชนีหลายชนิดบน HDFS เพื่อจัดเก็บข้อมูลการสำรวจทางไกลอย่างมีประสิทธิภาพ ดังนั้น งานวิจัยนี้ใช้ชุดข้อมูลที่อยู่ในรูปแบบวัตถุของจาวาสคริปต์ (JavaScript Object Notation, JSON ขนาดใหญ่ โดยใช้หลักการแบ่งข้อมูลและสตีมข้อมูลเข้าไปยังในหน่วยความจำร่วมกับการจัดทำดรรชนีเพื่อเข้าถึงข้อมูลบนเครื่องคอมพิวเตอร์ส่วนบุคคล อย่างไรก็ตาม งานวิจัยที่เกี่ยวข้องกับการสตรีมข้อมูลข้อมูลขนาดใหญ่และการจัดทำดรรชนี เช่น (Jiang และZhao, 2022) เสนอเทคนิคการสตรีมข้อมูลที่มีโครงสร้างกึ่งสมบูรณ์โดยใช้การเร่งด่วนแบบบิตขนาน (Belazzougui และคณะ, 2020) เสนอเทคนิคการสร้างดัชนีและวิเคราะห์สตริงในเวลาเชิงเส้นโดยใช้พื้นที่จัดเก็บข้อมูลน้อย (Chang และคณะ, 2017) เสนอเทคนิคการเพิ่มประสิทธิภาพของการค้นหาข้อมูลเมตาดาต้าในระบบไฟล์ขนาดใหญ่โดยการใช้วิธีดรรชนี บทความดังกล่าวสนับสนุนการเร่งความเร็วในการสตรีมและวิเคราะห์ข้อมูล JSON โดยการข้ามโครงสร้างข้อมูลที่ไม่เกี่ยวข้องกับการค้นหาและการจัดทำดัชนี นอกจากนี้ในส่วนบทที่ 3 ของการวิจัยเกี่ยวข้องกับการเพิ่มประสิทธิภาพดรรชนีด้วยเทคนิคการเข้าถึงเชิงเส้น (Linear search ,LS) การค้นหาแบบทวิภาค (Binary search, BS) ต้นไม้ค้นหาแบบทวิภาค (Binary search Tree, BST) และ (Adelson-Velsky and Landis Tree, AVL Tree) ซึ่งปัจจุบันมีงานวิจัยที่เกี่ยวของหลายบทความในการประยุกต์กับชุดข้อมูลขนาดใหญ่รวมถึงเพิ่มประสิทธิภาพด้านต่าง ๆ ของเทคนิคข้างต้น เช่น (Baykasoğlu และOzsoydan, 2018) เสนอเทคนิคการหาค่าเหมาะสมแบบไดนามิกในพื้นที่ค้นหาแบบไบนารี (Belazzougui และคณะ, 2020) เสนอเทคนิคการสร้างดัชนีและการวิเคราะห์สตริงในเวลาเชิงเส้นและใช้พื้นที่เล็กน้อย (Komorowski และTrzciński, 2019) เสนอเทคนิคการค้นหาเพื่อนบ้านที่ใกล้ที่สุดโดยประมาณโดยใช้ต้นไม้ค้นหาแบบไบนารีสุ่ม (Chopade และPachghare, 2020) เสนอเทคนิคการใช้ดัชนีใน MongoDB และ (Wang และคณะ, 2023) เสนอเทคนิคการเข้ารหัสที่สามารถค้นหาได้แบบมีการยืนยัน ซึ่งบทความดังกล่าวสนับสนุนการพัฒนาเทคนิคดรรชนีและค้นหาเข้าถึงข้อมูลอย่างประสิทธิภาพ และลดระยะเวลาในการจัดการข้อมูลขนาดใหญ่

บทที่ 3

วิธีดำเนินการวิจัย

การจัดทำดรรชนี (Indexing) สำหรับฐานข้อมูลนั้น ผู้วิจัยได้จำแนกการจัดทำดรรชนีออกเป็นสองประเภทหลัก ได้แก่ ดรรชนีแบบหนาแน่น (Dense Index) แบบหนึ่งต่อหนึ่ง (One-to-One) และดรรชนีแบบกระจาย (Sparse Index) แบบหนึ่งต่อกลุ่ม (One-to-May) โดยที่เทคนิคการจัดทำดรรชนี เหล่านี้ได้รับความนิยมอย่างแพร่หลายบนฐานข้อมูลเชิงสัมพันธ์ (Relational Database) อย่างไรก็ตาม ยังมีข้อจำกัดของการจัดทำดรรชนีข้อมูลกับฐานข้อมูลเชิงสัมพันธ์เพียงอย่างเดียว นอกจากนี้ เทคโนโลยีฐานข้อมูลไม่ใช่เชิงสัมพันธ์ (Not only Structured Query Language, NoSQL) ยังมีข้อดีทางด้านความยืดหยุ่นในการจัดเก็บข้อมูล มีความเร็วสูงต่อการสืบค้นข้อมูล อันเนื่องมาจากไม่มีโครงสร้างเชิงสัมพันธ์ ข้อจำกัดคือยากต่อการจัดทำดรรชนี สำหรับชุดข้อมูลไฟล์ JSON ขนาดใหญ่ ดังนั้นจึงต้องมีการทดสอบประสิทธิภาพโดยพิจารณาจากทางด้านเวลาของการเข้าถึงข้อมูลด้วยดรรชนี

งานวิจัยครั้งนี้เสนอการพัฒนาเทคนิคสร้างดรรชนีสำหรับข้อมูลขนาดใหญ่ที่อยู่ในรูปแบบวัตถุของจาวาสคริปต์ (The development of indexing techniques for large-scale data in the JavaScript object format) ผู้วิจัยได้จำลองข้อมูลไม่ต่ำกว่าแปดแสนรายการขึ้นไป ซึ่งมีขนาดไฟล์ข้อมูลไม่ต่ำกว่าสามสิบกิ๊กกะไบต์ ตามที่อธิบายไว้ในส่วนที่ 3.1 ภายใต้ชนิดไฟล์ (JavaScript Object Notation, JSON) ขนาดใหญ่ ซึ่งเป็นฐานข้อมูลไม่ใช่เชิงสัมพันธ์ (Not only Structured Query Language, NoSQL) จากนั้นนำข้อมูลไฟล์เข้าสู่กระบวนการวิเคราะห์ออกแบบขั้นตอนวิธีและดำเนินการวิจัย โดยผสมผสานบางเทคนิคของฐานข้อมูลเชิงสัมพันธ์ ได้แก่ การจัดทำดรรชนีแบบหนาแน่น (Dense Index) และการจัดทำดรรชนีแบบกระจาย (Sparse Index) รวมถึงประยุกต์ใช้โครงสร้างแบบแถวลำดับ (Array) การค้นหาแบบทวิภาค (Binary search) ต้นไม้ค้นหาแบบทวิภาค (Binary search Tree) และ Adelson-Velskii Landis Tree (AVL Tree) นอกจากนี้ ผู้วิจัยนำภาษาไพทอน (Python) สำหรับพัฒนาชุดคำสั่งและทดสอบเปรียบเทียบประสิทธิภาพเวลาการเข้าถึงข้อมูลของดรรชนี กับลักษณะคำหลักที่แตกต่างกันตามที่อธิบายไว้ในส่วนที่ 3.4 ดังรายละเอียดต่อไปนี้

3.1 ขั้นตอนการจำลองชุดข้อมูลขนาดใหญ่ในไฟล์รูปแบบวัตถุของจาวาสคริปต์

ในการวิจัยส่วนนี้ ได้จำลองชุดข้อมูลที่ชื่อว่า Bigdata.json ซึ่งประกอบด้วยข้อมูลขนาด 32 กิกะไบต์ และมีข้อมูล 1,000,000 รายการ ชุดไฟล์ข้อมูลจัดเก็บอยู่ในรูปแบบวัตถุของจาวาสคริปต์ (JavaScript Object Notation, JSON) และประกอบด้วยฟิลด์ที่หลากหลาย เช่น First name, Last name, Email และ Hex of image โดยชุดข้อมูลไฟล์ Bigdata.json ของผู้วิจัยมีความยืดหยุ่นรองรับการปรับแต่งเพื่อตอบสนองความต้องการของการวิจัยเฉพาะส่วนได้ ดังภาพ 18 แสดงตัวอย่างการทำข้อมูลบางรายการจากชุดข้อมูล ทั้งนี้ชุดไฟล์ Bigdata.json จะใช้ประกอบการอธิบายในหัวข้อ 3.1 ถึง 3.3 รวมถึงบทที่ 4


```
{
   "First_name": "Phillip",
   "Last_name": "Headlon",
   "Email": "Phillip.Headlon@smith.com",
   "Hex": "ffd8ffe000104a4649460001010100480048000ffdb004300..."
},
{
   "First_name": "Alice",
   "Last_name": "Wonder",
   "Email": "Alice.Wonder@example.net",
   "Hex of image": "ffd8ffe000104a464946000102020048004800000ffdb004300...
},
{
"........."
},
{
   "First_name": "Bob",
   "Last_name": " Marley",
   "Email": "Bob.Marley@gmail.org
   "Hex of image": "ffd8ffe000104a4649460001040400480048000ffdb004300..."
},
{
   "First_name": " Zoe",
   "Last_name": " Ziggler",
   "Email": " Zoe.Ziggler@yahoo.com",
   "Hex of image": "ffd8ffe000104a4649460001050500480048000ffdb004300..."
}
```

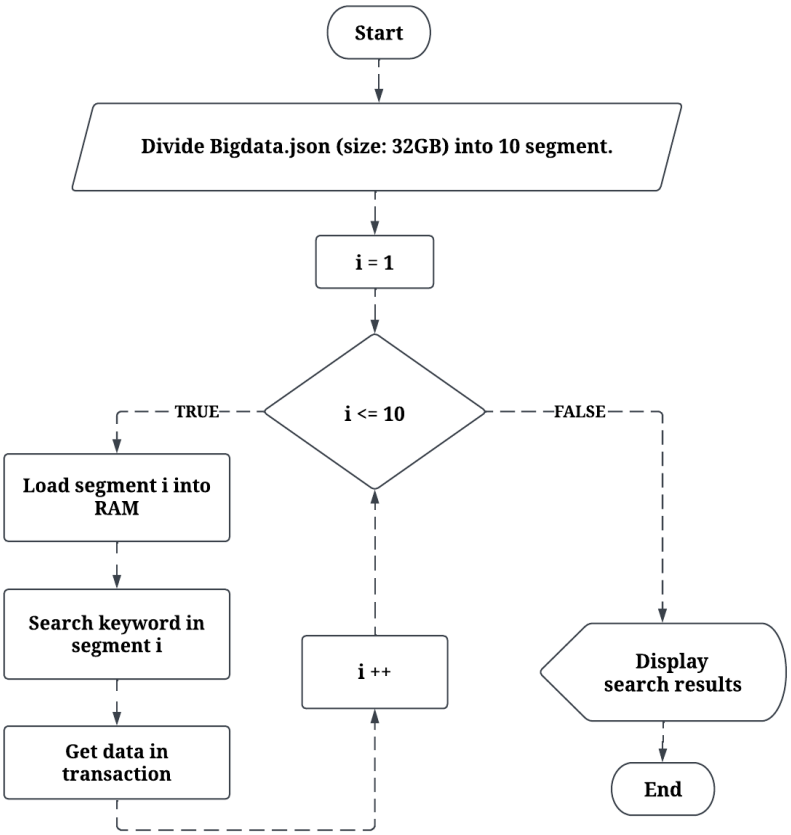

ภาพ 18 แสดงตัวอย่างของชุดไฟล์ข้อมูล Bigdata.json
สำหรับการสำรวจภายใต้รูปแบบ JSON ที่มีขนาด 32 GB และ 1,000,000 ข้อมูล

3.2 อัลกอริทึมสำหรับการดึงข้อมูลโดยไม่มีดรรชนี

เมื่อต้องจัดการกับชุดข้อมูลขนาดใหญ่ ภายใต้รูปแบบวัตถุของจาวาสคริปต์ (JavaScript Object Notation, JSON) ของฐานข้อมูลไม่ใช่เชิงสัมพันธ์ (Not only Structured Query Language, NoSQL) โดยไม่มีการจัดทำดรรชนีกับชุดข้อมูลที่เรียกว่า Bigdata.json ซึ่งมีขนาด 32 กิกะไบต์และมีข้อมูล 1,000,000 รายการ การศึกษานี้เมื่อต้องนำเข้าข้อมูลทั้งหมดไปยังหน่วยความจำเข้าถึงแบบสุ่ม (Random Access Memory, RAM) ทั้งหมดพร้อมกันเป็นสิ่งที่ท้าทายเป็นอย่างมาก ที่จะทำให้การเข้าถึงและค้นหาข้อมูลเกิดปัญหาขึ้นได้ การศึกษานี้นำเสนอเทคนิคในการเรียกสำรวจและเข้าถึงข้อมูล โดยไม่ต้องจัดทำดรรชนีเพื่อแก้ไขปัญหานี้ รหัสเทียมสำหรับการดึงข้อมูลโดยไม่มีดรรชนีแสดงดังภาพ 19 และแสดงด้วยผังงานดังภาพ 20 เทคนิคนี้เกี่ยวข้องกับการแบ่งข้อมูลจาก Bigdata.json ออกเป็นสิบส่วน แต่ละส่วนประกอบด้วย 3.2 กิกะไบต์ และ 100,000 รายการ ในระหว่างกระบวนการแบ่งกลุ่ม Bigdata.json ขั้นตอนนี้จะไม่เกี่ยวข้องกับการแยกไฟล์ออกเป็นสิบไฟล์แยกกันแต่จะเกี่ยวข้องกับการนำเข้าของการแบ่งส่วน (Segment size) ลงใน RAM หลังจากนั้น ระบบจะดึงข้อมูลโดยดึงข้อมูล แต่ละส่วนภายใน RAM หากข้อมูลตรงกับคำหลัก (Keyword) ที่กำหนด ระบบจะแยกและแสดงหลังจากดึงการแบ่งส่วน (Segment) ทั้งหมด การดึงข้อมูลครอบคลุมกลุ่ม Bigdata.json ทั้งหมด แม้ว่าเทคนิคนี้จะช่วยดึงและเข้าถึงข้อมูล แต่ก็มีข้อเสียที่สำคัญ เช่น การเรียกใช้ทรัพยากรของ RAM และต้องใช้พลังการประมวลผลจำนวนมากของหน่วยประมวลผลกลาง (Central Processing Unit, CPU) ส่งผลให้การสำรวจและเข้าถึงข้อมูลเกิดความล่าช้า

```
1. Import Bigdata.json and specify the keyword to search for.
2. Divide Bigdata.json (size of 32 GB) into 10 segments.
3. For each segment (i = 1 to 10):
    A. Load the segment i into RAM.
    B. Search for specified keyword in segment i.
    C. Retrieve the relevant data from segment i.
4. Display final search result
```

ภาพ 19 รหัสเทียมของการดึงข้อมูลโดยไม่มีดรรชนี



ภาพ 20 ผังงานของการดึงข้อมูลโดยไม่มีดรรชนี

3.3 อัลกอริทึมสำหรับการสำรวจและเข้าถึงข้อมูลโดยดรรชนี

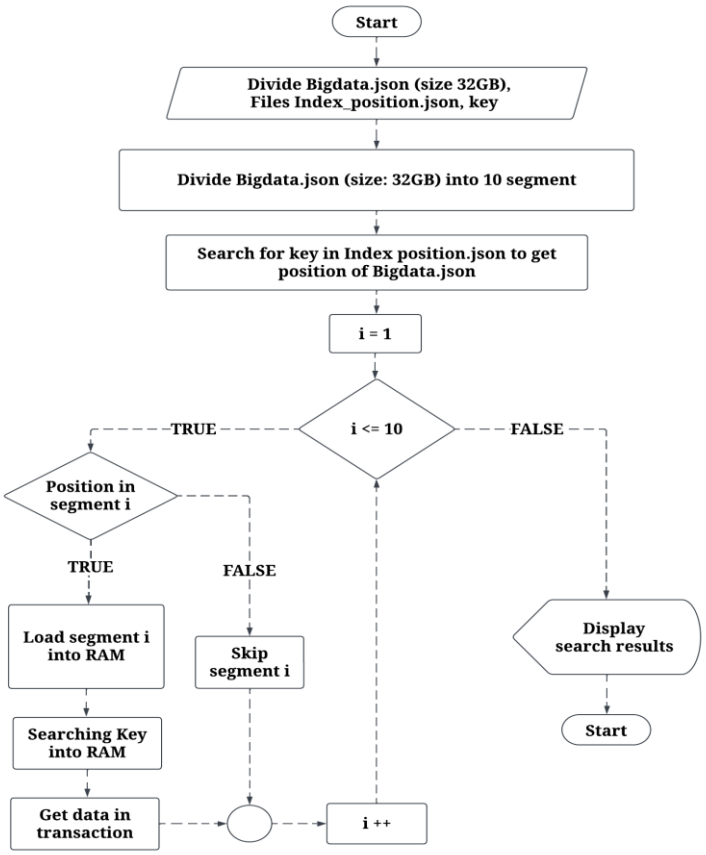
งานวิจัยส่วนนี้ เสนออัลกอริทึมสำหรับการเพิ่มประสิทธิภาพเวลาในการสำรวจและเข้าถึงข้อมูลของฐานข้อมูลไม่ใช่เชิงสัมพันธ์ (Not only Structured Query Language, NoSQL) ขนาดใหญ่ ภายใต้รูปแบบวัตถุของจาวาสคริปต์ (JavaScript Object Notation, JSON) โดยใช้ดรรชนีแบบหนาแน่น (Dense Index) และดรรชนีแบบกระจาย (Sparse index) จากฐานข้อมูลเชิงสัมพันธ์ (Relational databases) กับชุดไฟล์ข้อมูล Bigdata.json ดังภาพ 21 รหัสเทียมของการสำรวจและเข้าถึงข้อมูลจากดรรชนี และดังภาพ 22 แผนผังลำดับงานของการสำรวจและเข้าถึงชุดข้อมูลขนาดใหญ่ตามดรรชนี ซึ่งอัลกอริทึมได้มีการปรับให้เหมาะสม และแบ่งเป็นการจัดทำดรรชนีแบบหนาแน่น (Dense Index) และดรรชนีแบบกระจาย (Sparse index) สำหรับชุดข้อมูล NoSQL ซึ่งมีขนาด 32 กิกะไบต์ รวมถึงคำหลัก (Keyword) และตำแหน่ง (Position) เพื่อกำหนดสำหรับการอ้างอิงที่มีประสิทธิภาพ จากหน่วยความจำเข้าถึงแบบสุ่ม (Random Access

Memory, RAM) ไปยังชุดไฟล์ข้อมูล Bigdata.json ระหว่างกระบวนการการสำรวจด้วยเทคนิคการเข้าถึงข้อมูลตามคำหลัก (Keyword-Index) ชุดคำสั่งจะดำเนินการนำเข้าชุดข้อมูล Bigdata.json ที่มีขนาด 32 กิกะไบต์ และมีข้อมูล 1,000,000 รายการ ซึ่งชุดข้อมูลไฟล์ Bigdata.json จะถูกแบ่งออกเป็น 10 ส่วน แต่ละส่วนประกอบด้วยข้อมูล 100,000 รายการ ได้กำหนดส่วนแรกให้สอดคล้องกับตำแหน่งของข้อมูล 1 ถึง 100,000 ส่วนที่สองเป็นตำแหน่งของข้อมูล 100,001 ถึง 200,000 และดำเนินต่อไป จนถึงส่วนที่สิบแสดงตำแหน่ง 900,001 ถึง 1,000,000 ถัดไป อัลกอริทึมจะตรวจสอบตำแหน่งของข้อมูล ในแต่ละส่วนเทียบกับไฟล์ดรรชนี หากตำแหน่งอยู่ในของการแบ่งส่วน (Segment) อัลกอริทึมจะนำเข้าส่วน (Segment) ลงในหน่วยความจำ และเข้าถึงข้อมูลโดยใช้คำหลัก (Keyword) เพื่อค้นหาข้อมูลที่ตรงกับคำหลัก (Keyword) ที่กำหนด เมื่อพบการจับคู่ที่ตรงกันอัลกอริทึมจะเข้าถึงข้อมูล และแสดงข้อมูลภายในกลุ่มตามผลลัพธ์ที่พบ

```
1. Import files index_position.json, Bigdata.json and Keyword
2. Divide Bigdata.json (size of 32 GB) into 10 segments.
3. Search for key in index position.json to get position of Bigdata.json
4. For segment i = 1 to 10
    A. If position is in segment i.
        A1. Load segment i into RAM
        A2. Search position in segment i
        A3. Retrieve the relevant data from segment i.
    B. Else, skip segment i.
5. Display search result
```

ภาพ 21 รหัสเทียมของการสำรวจและเข้าถึงข้อมูลจากดรรชนี

ขนาดของกลุ่มที่ใช้ในอัลกอริทึมอาจมีการเปลี่ยนแปลงตามความเหมาะสมสำหรับชุดข้อมูลที่กำหนดในรหัสเทียมที่แสดงเป็นตัวอย่างของการกำหนดค่าดังกล่าว โดยสมมติว่าการแบ่งส่วน (Segment) มีขนาด 100,000 ข้อมูล อย่างไรก็ตาม ขนาดส่วนที่เหมาะสมที่สุดที่กำหนดโดยการทดลองจะถูกรายงานในตาราง 6 และ 7 เพื่อให้แน่ใจว่าผลการวิจัยมีความถูกต้อง ได้จัดเตรียมวิธีการโดยละเอียดและข้อมูลอ้างอิงที่สนับสนุนแนวทางของผู้วิจัย



ภาพ 22 แผนผังลำดับงานของการสำรวจและเข้าถึงชุดข้อมูลขนาดใหญ่ตามดรรชนี

3.3.1 ดรรชนีแบบหนาแน่นสำหรับชุดข้อมูลขนาดใหญ่

ในการวิจัยส่วนนี้ ได้เพิ่มประสิทธิภาพการสำรวจและเข้าถึงข้อมูลแบบหนึ่งต่อหนึ่ง (One-to-One) ในชุดข้อมูลขนาดใหญ่ ซึ่งเก็บอยู่ในรูปแบบวัตถุของจาวาสคริปต์ (JavaScript Object Notation, JSON) บนฐานข้อมูลไม่ใช่เชิงสัมพันธ์ (Not only Structured Query Language, NoSQL) การเชื่อมโยงข้อมูลแบบหนึ่งต่อหนึ่ง (One-to-One) ในที่นี้หมายถึงการเชื่อมโยงความสัมพันธ์เฉพาะคำหลัก (Keyword) ซึ่งเป็นสิ่งที่ท้าทายเป็นอย่างมาก เมื่อต้องจัดการกับข้อมูลขนาดใหญ่ ได้เสนออัลกอริทึม (Algorithm) ที่เรียกว่า การสร้างดรรชนีแบบหนาแน่น (Dense Index Algorithm) เป็นอัลกอริทึมที่ได้รับการออกแบบมา เพื่อเพิ่มประสิทธิภาพในเชิงเวลาของการสำรวจและเข้าถึงข้อมูลขนาดใหญ่ ส่วนถัดไปของงานวิจัยนี้ จะเป็นการให้คำอธิบายที่ครอบคลุมถึงอัลกอริทึมในการสร้างดรรชนีแบบหนาแน่น รวมถึงวิธีการที่ไฟล์ดรรชนีแบบหนาแน่น ถูกนำไปใช้ในการสำรวจและเข้าถึงชุดข้อมูลขนาดใหญ่

3.3.1.1 อัลกอริทึมการสร้างตำแหน่งของดรรชนีแบบหนาแน่น สำหรับชุดข้อมูลขนาดใหญ่

ในการวิจัยส่วนนี้ คำว่า ชุดข้อมูลขนาดใหญ่ (Large-scale dataset) หมายถึงไฟล์ชุดการจำลองที่ชื่อว่า Bigdata.json การวิจัยนี้ “ดรรชนีความหนาแน่น” คือไฟล์ที่มีชื่อว่า Dense_index_position.json ซึ่งสร้างโดยการสแกนไฟล์ Bigdata.json ครั้งเดียว เพื่อดึงข้อมูลที่ไม่ซ้ำซ้อนในฟิลด์ที่เลือกเป็นคำหลัก (Keyword) การศึกษานี้ประกอบอีเมล์ (Email) และตำแหน่ง (Position) ที่สอดคล้องกันในไฟล์ Bigdata.json ไฟล์ดรรชนีแบบหนาแน่นของผู้วิจัยมีโครงสร้างที่แสดงไว้ดังภาพ 23 เป็นการออกแบบเพื่อการอ้างอิงแบบหนึ่งต่อหนึ่ง (One to One) และการเข้าถึงข้อมูลได้อย่างมีประสิทธิภาพ ภายในโครงสร้างของไฟล์ตำแหน่งดรรชนีแบบหนาแน่นประกอบด้วยข้อมูลอีเมล์ (Email) และตำแหน่ง (Position) โดยตำแหน่งที่สร้างขึ้นนั้นสามารถช่วยในการเข้าถึงข้อมูลจากชุดไฟล์ Bigdata.json ที่เก็บไว้ในหน่วยความจำเข้าถึงแบบสุ่ม (Random Access Memory, RAM) ได้อย่างรวดเร็ว การศึกษาไฟล์ตำแหน่งดรรชนีแบบหนาแน่นของผู้วิจัยมีขนาด 20 เมกะไบต์ และมีข้อมูล 1,000,000 รายการ ทั้งนี้ Dense_index_position.json จะใช้ประกอบการอธิบายในหัวข้อ 3.3.1.2

```
{
 "Phillip.Headlon@smith.com": 0,
 "Alice.Wonder@example.net": 1,
 "Bob.Marley@gmail.org": 2,
 ...
 "Zoe.Ziggler@yahoo.com": 999999
}
```

ภาพ 23 แสดงตัวอย่างข้อมูลในชุดไฟล์ Dense index position.json

3.3.1.2 อัลกอริทึมการใช้งาน Dense_index_position.json เพื่อสำรวจและเข้าถึงชุดข้อมูลขนาดใหญ่

สำหรับชุดไฟล์ดรรชนีแบบหนาแน่น หรือ Dense_index_position.json มีความสามารถในการสำรวจและเข้าถึงฐานข้อมูลไม่ใช่เชิงสัมพันธ์ (Not only Structured Query Language, NoSQL) ขนาดใหญ่ ซึ่งใช้รูปแบบวัตถุของจาวาสคริปต์ (JavaScript Object Notation, JSON) การสร้างการจัดทำตำแหน่งของดรรชนีแบบหนาแน่นนี้ เริ่มต้นด้วยการนำเข้าไฟล์ Dense_index_position.json และ Bigdata.json โดยมีการกำหนดอีเมล (Email) เป็นคำหลัก (Keyword) สำหรับการสำรวจและเข้าถึงข้อมูล การดำเนินการดังกล่าวทำให้สามารถเรียก

ข้อมูลที่สอดคล้องกับตำแหน่งของคำหลักได้อย่างรวดเร็ว ด้วยขนาดไฟล์ที่ 20 เมกะไบต์ ส่งผลให้มีประสิทธิภาพสูงในการเข้าถึงข้อมูล อย่างไรก็ตาม เมื่อตำแหน่งที่ทราบสอดคล้องกับคำหลักเพียงตำแหน่งเดียวภายในไฟล์ จำเป็นต้องนำเข้าเฉพาะส่วนที่เกี่ยวข้องของไฟล์ Dense_index_position.json ลงในหน่วยความจำเข้าถึงแบบสุ่ม (Random Access Memory, RAM) เพื่อการเข้าถึงข้อมูล ในกรณีที่คำหลักไม่ตรงกับข้อมูลใด ๆ ในชุดไฟล์ จะไม่ได้รับตำแหน่งใด ๆ และจึงไม่มีการนำเข้าส่วนใดของไฟล์ Dense_index_position.json ลงในหน่วยความจำ

3.3.2 ดรรชนีแบบกระจาย สำหรับชุดข้อมูลขนาดใหญ่

ในการวิจัยส่วนนี้ ได้เพิ่มประสิทธิภาพในการสำรวจและเข้าถึงข้อมูลแบบหนึ่งต่อกลุ่ม (One-to-May) ในชุดข้อมูลขนาดใหญ่ ซึ่งเก็บอยู่ในรูปแบบวัตถุของจาวาสคริปต์ (JavaScript Object Notation, JSON) บนฐานข้อมูลไม่ใช่เชิงสัมพันธ์ (Not only Structured Query Language, NoSQL) การเชื่อมโยงข้อมูลแบบหนึ่งต่อกลุ่ม (One-to-Many) ในที่นี้หมายถึงความสัมพันธ์เฉพาะคำหลัก (Keyword) สามารถเชื่อมโยงกับกลุ่มข้อมูลคำหลักที่เหมือนกัน ซึ่งเป็นสิ่งที่ท้าทายเป็นอย่างมาก เมื่อต้องจัดการกับข้อมูลขนาดใหญ่ ได้เสนออัลกอริทึม (Algorithm) ที่เรียกว่า ดรรชนีแบบกระจาย (Sparse Index Algorithm) เป็นอัลกอริทึมที่ได้รับการออกแบบมา เพื่อเพิ่มประสิทธิภาพในเชิงเวลาของการสำรวจและเข้าถึงข้อมูลขนาดใหญ่ ส่วนถัดไปของงานวิจัยนี้ จะเป็นการให้คำอธิบายที่ครอบคลุมถึงอัลกอริทึมในการสร้างดรรชนีแบบกระจาย รวมถึงวิธีการที่ไฟล์ดรรชนีแบบกระจาย ถูกนำไปใช้ในการสำรวจและเข้าถึงชุดข้อมูลขนาดใหญ่

3.3.2.1 อัลกอริทึมการสร้างตำแหน่งของดรรชนีแบบกระจาย สำหรับชุดข้อมูลขนาดใหญ่

การวิจัยส่วนนี้ ให้คำจำกัดความของชุดข้อมูลขนาดใหญ่เป็นไฟล์ Bigdata.json ได้ทำการออกแบบและพัฒนาชุดคำสั่งสร้างไฟล์ดรรชนีแบบกระจาย (Sparse Index) ที่เรียกว่า Sparse_index_position.json จากชุดข้อมูลไฟล์ Bigdata.json โดยได้มาจากการสแกนหนึ่งครั้งเพื่อแยกข้อมูลที่ไม่ซ้ำกัน โดยเลือกใช้ชื่อแหล่งที่มา (Resource_name) เป็นฟิลด์คำหลัก (Keyword) ชุดไฟล์ดรรชนีแบบกระจาย (Sparse Index) นี้ แสดงดังภาพ 24 ได้ออกแบบและพัฒนาเพื่อเพิ่มประสิทธิภาพเชิงเวลาในการสำรวจและเข้าถึงข้อมูลแบบหนึ่งต่อกลุ่ม (One-to-Many) โดยจะมีทั้งชื่อแหล่งที่มา (Resource_name) และข้อมูลตำแหน่ง (Position) ที่เกี่ยวข้องโดยที่ชื่อแหล่งที่มา (Resource_name) อาจมีหลายตำแหน่ง (Position)

ข้อมูลในไฟล์ดรรชนีของผู้วิจัยสามารถช่วยให้การอ้างอิงจากหน่วยความจำเข้าถึงแบบสุ่ม (Random Access Memory, RAM) ไปยังข้อมูลภายในชุดไฟล์ Bigdata.json ได้อย่างมีประสิทธิภาพเชิงเวลา เมื่อมีการเรียกใช้งานชุดไฟล์ดรรชนีแบบกระจาย (Sparse Index) โปรดทราบว่าไฟล์ดรรชนีแบบกระจาย (Sparse Index) ที่เรียกว่า Sparse_index_position.json ของผู้วิจัยมีขนาดไฟล์ 8 เมกะไบต์ และมีข้อมูล 5,146 รายการ

```
{
   " Phillip": [46163, 66867, 88916, 94633, 130606, 150253, 185197, 189917, 214931, 999999],
   ……
   " Alice": [46153, 59781, 93151, 94460, 101100, 172271, 244064, 368397, 439620, 888888]
}
```

ภาพ 24 แสดงตัวอย่างข้อมูลในชุดไฟล์ Sparse index position.json

3.3.2.2 อัลกอริทึมการใช้งาน Sparse_index_position.json เพื่อสำรวจและเข้าถึงชุดข้อมูลขนาดใหญ่

ในการสำรวจและเข้าถึงข้อมูลรูปแบบวัตถุของจาวาสคริปต์ (JavaScript Object Notation, JSON) บนฐานข้อมูลไม่ใช่เชิงสัมพันธ์ (Not only Structured Query Language, NoSQL) ขนาดใหญ่ สามารถใช้ไฟล์ดรรชนีแบบกระจาย (Sparse_index_position.json) แสดงดังภาพ 24 ไฟล์ดรรชนีนี้เหมาะอย่างยิ่งสำหรับการสำรวจและการเข้าถึงแบบหนึ่งต่อกลุ่ม (One-to-Many) โดยเริ่มจากการนำเข้าข้อมูล ไฟล์ดรรชนีแบบกระจาย (Sparse_index_position.json) ประกอบด้วยชื่อแหล่งที่มา (Resource_name) และตำแหน่ง (Position) ที่สอดคล้องกัน ซึ่งทำหน้าที่เป็นจุดอ้างอิงไปยังข้อมูลภายในไฟล์ดรรชนีแบบกระจาย (Sparse_index_position.json) ที่จัดเก็บไว้ในหน่วยความจำเข้าถึงแบบสุ่ม (Random Access Memory, RAM) ซึ่งนำไปสู่เวลาในการค้นหาที่มีประสิทธิภาพ ส่วนภาพ 18 แสดงกระบวนการที่เริ่มต้นด้วยการนำเข้าไฟล์ Bigdata.json ในบริบทนี้ ชื่อแหล่งที่มา (Resource_name) คือคำหลักสำหรับการสำรวจและเข้าถึงข้อมูล เมื่อดำเนินการเรียกข้อมูลโดยใช้ไฟล์ (Sparse_index_position.json) คำสั่งจะเข้าถึงตำแหน่งที่เกี่ยวข้องกับคำหลัก (Keyword) ที่กำหนด โปรดทราบว่าดรรชนีแบบกระจาย (Sparse_index_position.json) ของผู้วิจัยมีขนาดค่อนข้างเล็กเพียง 8 เมกะไบต์ ซึ่งมีส่วนช่วยให้ประสิทธิภาพโดดเด่นในการเข้าถึงข้อมูลอย่างรวดเร็ว อย่างไรก็ตาม การเข้าถึงคำหลัก (Keyword) เมื่อทราบตำแหน่ง (Position) หลายตำแหน่งจากภายในไฟล์ดรรชนีแบบกระจาย (Sparse_index_position.json) ในกรณีเช่นนี้

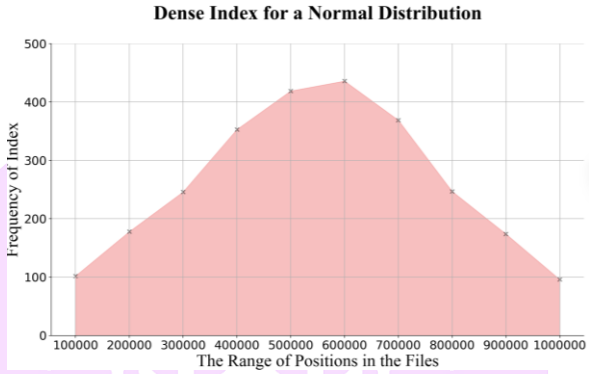
ระบบต้องนำเข้าดรรชนีหลายส่วน (Segment) ลงในหน่วยความจำ เพื่อแยกข้อมูลที่จำเป็น ในทางกลับกัน หากคำหลัก (Keyword) ไม่ตรงกับข้อมูลใด ๆ ในไฟล์ดรรชนีแบบกระจาย (Sparse_index_position.json) ชุดคำสั่งจะไม่เข้าถึงตำแหน่งนั้น ๆ และเป็นผลให้หน่วยความจำไม่ต้องนำเข้าส่วนดรรชนีใด ๆ

ในการดำเนินการค้นหาเพื่อเข้าถึงตำแหน่งในดรรชนีแบบหนาแน่น (Dense Index) และดรรชนีแบบกระจาย (Sparse Index) ไม่มีปัญหาเกิดขึ้น เมื่อค้นหาคำหลักคำเดียว อย่างไรก็ตาม เมื่อค้นหาด้วยคำหลักหลายคำ ควรพิจารณาลักษณะการกระจายของคำหลักที่แตกต่างกันในช่วงตำแหน่งภายในฐานข้อมูล ดังนั้น ส่วนถัดไปนี้ เสนอวิธีการเปรียบเทียบประสิทธิภาพเวลาของการเข้าถึงตำแหน่งในดรรชนีแบบหนาแน่นและดรรชนีแบบกระจายตามอธิบายไว้ในส่วนที่ 3.4 โดยที่ดรรชนีแบบหนาแน่นและดรรชนีแบบกระจาย ประกอบด้วยรูปแบบการกระจายคำหลักที่แตกต่างกันทั้งสามรูปแบบ ได้แก่ การกระจายแบบปกติ การกระจายที่มีลักษณะเอียงซ้าย และการกระจายที่มีลักษณะเอียงขวา เพื่อใช้สำหรับทดสอบประสิทธิภาพของการค้นหาเพื่อเข้าถึงตำแหน่งข้อมูลโดยใช้เทคนิคตามอธิบายไว้ในส่วนที่ 3.5 ประกอบด้วย การค้นหาเชิงเส้น (Linear search, LS) การค้นหาแบบทวิภาค (Binary search, BS) ต้นไม้ค้นหาแบบทวิภาค (Binary search Tree, BST) และ Adelson–Velskii Landis Tree (AVL Tree) วัตถุประสงค์ของการวิจัยนี้ คือการเปรียบเทียบประสิทธิภาพเวลาในการเข้าถึงตำแหน่งในดรรชนีทั้งสอง โดยอ้างอิงถึงรูปแบบการกระจายคำหลักที่แตกต่างกันและทดสอบประสิทธิภาพการค้นหาโดยใช้ทั้งสี่เทคนิค เพื่อพิจารณาว่าวิธีใดให้ผลลัพธ์ที่ดีที่สุดและช้าที่สุดในการเข้าถึงตำแหน่งในดรรชนีทั้งสอง

3.4 การเลือกดรรชนีคำหลักสำหรับดรรชนีแบบหนาแน่นและดรรชนีแบบกระจาย

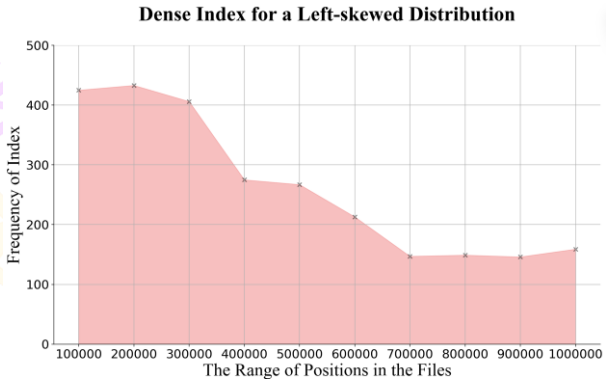
ส่วนนี้อธิบายการเลือกคำหลัก 524 คำเพื่อประเมินประสิทธิภาพของการวิจัย คำหลักเหล่านี้ จากไฟล์ตำแหน่งดรรชนีแบบหนาแน่น (Dense Index) แสดงดังภาพ 23 ซึ่งมีขนาดไฟล์เพียง 32 เมกะไบต์และมี 1,000,000 รายการ และไฟล์ตำแหน่งดรรชนีแบบกระจาย (Sparse Index) แสดงดังภาพ 24 ซึ่งมีขนาดไฟล์เพียง 8 เมกะไบต์และมี 5,146 รายการ สำหรับดรรชนีแบบหนาแน่น (Dense Index) ผู้วิจัยคัดเลือกคำหลักของดรรชนีจากข้อมูลละติจูด (Latitude) ลองจิจูด (Longitude) และตำแหน่ง (Position) เดียวของชุดข้อมูล Bigdata.json ออกเป็นห้าไฟล์ แต่ละไฟล์มีคำหลักที่ไม่ซ้ำ 524 คำ และแบ่งไฟล์ดรรชนีเหล่านี้ออกเป็นสามประเภทหลักตามช่วงตำแหน่ง ประเภทแรก คือดรรชนีแบบหนาแน่นที่มีการแจกแจงแบบปกติ () ซึ่งมีคำหลักทั้งหมด 2,070 คำ แจกแจงอยู่ในช่วงตำแหน่ง 300,000 ถึง

800,000 และ 550 คำหลัก แจกแจงอยู่ทางด้านซ้ายและทางด้านขวา แสดงดังภาพ 27 ทั้งนี้ การเลือกดรรชนีคำหลักสำหรับดรรชนีแบบหนาแน่นและดรรชนีแบบกระจายจะใช้ประกอบการอธิบายในหัวข้อ 3.5



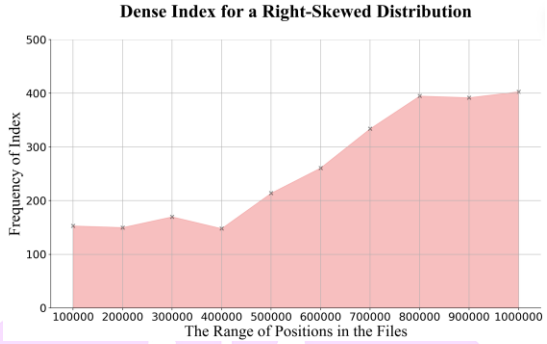
ภาพ 25 ดรรชนีแบบหนาแน่น สำหรับการแจกแจงปกติ

ดรรชนีประเภทที่สอง คือดรรชนีแบบหนาแน่น สำหรับการแจกแจงที่มีลักษณะเอียงซ้าย ซึ่งมีคำหลักทั้งหมด 1,806 คำ แจกแจงอยู่ในช่วงตำแหน่ง 100,000 ถึง 500,000 และ 814 คำหลัก แจกแจงอยู่ทางด้านขวา แสดงดังภาพ 28



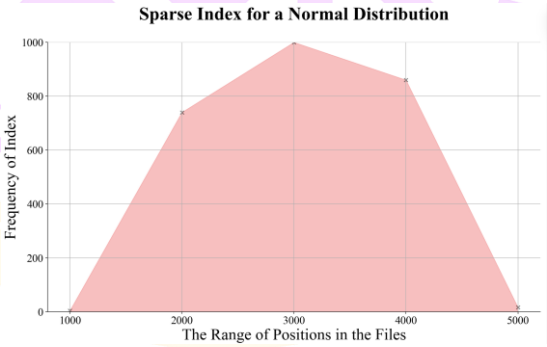
ภาพ 26 ดรรชนีแบบหนาแน่น สำหรับการแจกแจงที่มีลักษณะเอียงซ้าย

ดรรชนีประเภทที่สาม คือดรรชนีแบบหนาแน่น สำหรับการแจกแจงที่มีลักษณะเอียงขวา ซึ่งมีคำหลักทั้งหมด 1,999 คำ แจกแจงอยู่ในช่วงตำแหน่ง 500,000 ถึง 1,000,000 และ 621 คำหลัก แจกแจงทางด้านซ้าย แสดงดังภาพ 29



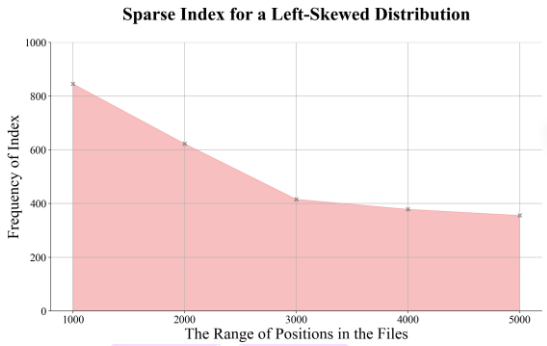
ภาพ 27 ดรรชนีแบบหนาแน่น สำหรับการแจกแจงที่มีลักษณะเอียงขวา

สำหรับดรรชนีแบบกระจาย (Sparse Index) ผู้วิจัยได้สร้างไฟล์ดรรชนีจากชื่อแหล่งที่มา (Resource_name) และข้อมูลตำแหน่ง (Position) ที่เกี่ยวข้องโดยที่ชื่อแหล่งที่มา (Resource_name) อาจมีหลายตำแหน่ง (Position) ออกเป็นห้าไฟล์ แต่ละไฟล์มีคำหลักที่ไม่ซ้ำกัน 524 คำ และแบ่งไฟล์ดรรชนีเหล่านี้ออกเป็นสามประเภทหลักตามช่วงตำแหน่ง ประเภทแรกคือ ดรรชนีแบบกระจาย สำหรับการแจกแจงปกติ ซึ่งมีคำหลัก 2,600 คำ แจกแจงอยู่ในช่วงตำแหน่ง 2,000 ถึง 4,000 และ 20 คำหลัก แจกแจงอยู่ทางด้านซ้ายและทางด้านขวา แสดงดังภาพ 30



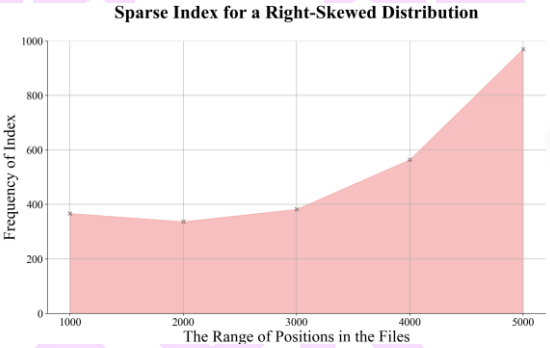
ภาพ 28 ดรรชนีแบบกระจาย สำหรับการแจกแจงปกติ

ดรรชนีประเภทที่สอง คือดรรชนีแบบกระจาย สำหรับการแจกแจงที่มีลักษณะเอียงซ้าย ซึ่งมีคำหลักทั้งหมด 1,885 คำ แจกแจงอยู่ในช่วงตำแหน่ง 1,000 ถึง 3,000 และ 735 คำหลัก แจกแจงอยู่ทางด้านขวา แสดงดังภาพ 31



ภาพ 29 ดรรชนีแบบกระจาย สำหรับการแจกแจงที่มีลักษณะเอียงซ้าย

ดรรชนีประเภทที่สาม คือดรรชนีแบบกระจาย สำหรับการแจกแจงที่มีลักษณะเอียงขวา ซึ่งมีคำหลักทั้งหมด 1,916 คำ แจกแจงอยู่ในช่วงตำแหน่ง 3,000 ถึง 5,000 และ 704 คำหลักแจกแจงอยู่ทางด้านซ้าย แสดงดังภาพ 32



ภาพ 30 ดรรชนีแบบกระจาย สำหรับการแจกแจงที่มีลักษณะเอียงขวา

3.5 อัลกอริทึมสำหรับการเปรียบเทียบประสิทธิภาพของดรรชนีแบบหนาแน่นและดรรชนีแบบกระจาย

ส่วนนี้อธิบายวิธีการค้นหาและเข้าถึงไฟล์ตำแหน่งดรรชนีแบบหนาแน่นและดรรชนีแบบกระจาย ซึ่งประกอบด้วย 3.5.1 อัลกอริทึมดรรชนีแบบหนาแน่นและดรรชนีแบบกระจาย โดยใช้เทคนิคการค้นหาเชิงเส้น (Linear search, LS) 3.5.2 อัลกอริทึมดรรชนีแบบหนาแน่นและดรรชนีแบบกระจาย โดยใช้เทคนิคการค้นหาแบบทวิภาค (Binary search, BS) 3.5.3 อัลกอริทึมดรรชนีแบบหนาแน่นและดรรชนีแบบกระจาย โดยใช้เทคนิคต้นไม้ค้นหาแบบทวิภาค (Binary search Tree, BST) และ 3.5.4 อัลกอริทึมดรรชนีแบบหนาแน่นและดรรชนีแบบกระจาย โดยใช้เทคนิค Adelson-Velskii Landis Tree (AVL Tree) รายละเอียดของอัลกอริทึมดังนี้

3.5.1 อัลกอริทึมดรรชนีแบบหนาแน่นและดรรชนีแบบกระจาย โดยใช้เทคนิคการค้นหาเชิงเส้น (Linear search, LS)

สำหรับวิธีการค้นหาตำแหน่งของคำหลักภายในไฟล์ดรรชนีแบบหนาแน่น (Dense Index) และดรรชนีแบบกระจาย (Sparse Index) โดยใช้เทคนิคการค้นหาเชิงเส้น (Linear search, LS) แสดงดังอัลกอริทึม 1 นำเข้าคำหลักทดสอบที่มีการแจกแจงที่แตกต่างกัน ซึ่งประกอบด้วยการแจกแจงแบบปกติ แบบเอียงขวา และแบบเอียงซ้าย สำหรับกระบวนการค้นหาเริ่มต้นด้วยการตรวจสอบคำหลักในแต่ละรายการภายในไฟล์ดรรชนีทั้งสอง หากคำหลักที่ต้องการสอดคล้องกับคำหลักที่มีอยู่ในไฟล์ ตำแหน่งของคำหลักนั้นจะสามารถเข้าถึงตำแหน่งข้อมูลได้ อย่างไรก็ตาม หากคำหลักไม่สอดคล้องกัน การค้นหาจะดำเนินการต่อไป โดยการตรวจสอบคำหลักในแต่ละรายการจนคำหลักสอดคล้องกัน หรือตรวจสอบรายการครบทั้งหมด

อัลกอริทึม 1. ดรรชนีแบบหนาแน่นและดรรชนีแบบกระจาย โดยเทคนิคที่เรียกว่า Linear Search (LS)

Input: Dense and Sparse Index position File, keywords test with distribution as described in section 3.1
Output: Position in Bigdata.json

```
def find_keyword(index_positions, desired_keyword):
    result = {}
    for keyword in desired_keyword:
        position_found = False
     for index, index_entry in enumerate(index_positions):
            if index_entry['keyword'] == keyword:
                result[keyword] = index_entry['position']
                position_found = True
                break
        if not position_found:
            result[keyword] = "Keyword not found."
    return result
```

การใช้อัลกอริทึมดรรชนีแบบหนาแน่น และดรรชนีแบบกระจาย ร่วมกับเทคนิคการค้นหาเชิงเส้น (Linear Search, LS) ในการระบุตำแหน่งของคำหลักในไฟล์ได้ถูกดำเนินการโดยใช้ชุดของคำหลักทดสอบที่มีรูปแบบการแจกแจงที่แตกต่างกัน ได้แก่ การแจกแจงแบบปกติ แบบเอียงขวา และแบบเอียงซ้าย ผลการค้นหาแต่ละครั้งเผยให้เห็นถึงประสิทธิภาพเวลาที่แปรผันตามจำนวนของรายการในไฟล์ดรรชนี ซึ่งปฏิบัติตามหลักความซับซ้อนเชิงเส้น $O(n)$ ในกรณีที่คำหลักตรงกับข้อมูลภายในไฟล์ดรรชนี ตำแหน่งของข้อมูลที่สอดคล้องกับ 'Bigdata.json' จะถูกคืนค่ากลับไปอย่างมีประสิทธิภาพ

3.5.2 อัลกอริทึมดรรชนีแบบหนาแน่นและดรรชนีแบบกระจาย โดยใช้เทคนิคการค้นหาแบบทวิภาค (Binary search, BS)

สำหรับวิธีการค้นหาตำแหน่งของคำหลักภายในไฟล์ดรรชนีแบบหนาแน่น (Dense Index) และดรรชนีแบบกระจาย (Sparse Index) โดยใช้การค้นหาแบบทวิภาค (Binary Search, BS) แสดงดังอัลกอริทึม 2 นำเข้าคำหลักทดสอบที่มีการแจกแจงที่แตกต่างกันซึ่งประกอบด้วยการแจกแจงแบบปกติ แบบเอียงขวา และแบบเอียงซ้าย ซึ่งไฟล์ดรรชนีทั้งสองจะถูกจัดเรียงตามตัวอักษร และใช้คำหลักทดสอบเหล่านี้ในการหาค่ากลางของลำดับคำหลัก โดยที่ค่ากลางนี้จะถูกเปรียบเทียบกับคำหลักที่ต้องการค้นหา หากพบคำหลักที่ต้องการ ตำแหน่งของคำหลักนั้นจะถูกส่งกลับ หากไม่พบ ข้อมูลจะถูกแบ่งออกเป็นสองส่วน และการค้นหาจะต่อเนื่องในส่วนที่มีคำหลักที่ใกล้เคียงกับคำหลักที่ต้องการมากที่สุด

อัลกอริทึม 2. ดรรชนีแบบหนาแน่นและดรรชนีแบบกระจาย โดยใช้เทคนิคเรียกว่า Binary Search (BS)

Input: Dense and Sparse Index position File, keywords test with distribution as described in section 3.1
Output: Position in Bigdata.json

```
def binary_search(index_file, desired_keyword):
    low, high = 0, len(index_file) - 1
    while low <= high:
        mid = (high + low) // 2
        mid_keyword = index_file[mid][0]
        if mid_keyword == desired_keyword:
            return index_file[mid][1]
        elif mid_keyword < desired_keyword:
            low = mid + 1
        else:
            high = mid - 1
    return "Keyword not found."
```

การใช้อัลกอริทึมดรรชนีแบบหนาแน่น และดรรชนีแบบกระจาย ร่วมกับเทคนิคการค้นหาแบบทวิภาค (Binary Search, BS) เพื่อเข้าถึงตำแหน่งของคำหลักในไฟล์ดรรชนีทั้งสองประเภท โดยนำเข้าคำหลักทดสอบที่มีรูปแบบการแจกแจงที่แตกต่างกัน ได้แก่ การแจกแจงแบบปกติ แบบเอียงขวา และแบบเอียงซ้าย กระบวนการตรวจสอบการค้นหาแต่ละครั้งจะใช้เทคนิคการค้นหาแบบทวิภาค ซึ่งประสิทธิภาพในการเข้าถึงตำแหน่งคำหลักขึ้นอยู่กับช่วงข้อมูล ในกรณีเฉลี่ยหรือกรณีที่ช้าที่สุด ความซับซ้อนของเวลาคือ $O(log\ n)$ อย่างไรก็ตาม ในสถานการณ์ที่ดีที่สุด ความซับซ้อนของเวลาอาจเป็น $O(1)$ เมื่อคำหลักทดสอบตรงกันอย่างสมบูรณ์กับไฟล์ตำแหน่งดรรชนี ซึ่งจะทำให้สามารถส่งคืนตำแหน่งของไฟล์ 'Bigdata.json'

3.5.3 อัลกอริทึมดรรชนีแบบหนาแน่นและดรรชนีแบบกระจาย โดยใช้เทคนิคต้นไม้ค้นหาแบบทวิภาค (Binary search Tree, BST)

สำหรับวิธีการค้นหาตำแหน่งของคำหลักภายในไฟล์ดรรชนีแบบหนาแน่น (Dense Index) และดรรชนีแบบกระจาย (Sparse Index) โดยใช้เทคนิคต้นไม้ค้นหาแบบทวิภาค (Binary search Tree, BST) แสดงดังอัลกอริทึม 3 นำเข้าคำหลักทดสอบที่มีการแจกแจงที่แตกต่างกันซึ่งประกอบด้วยการแจกแจงแบบปกติ แบบเอียงขวา และแบบเอียงซ้าย กระบวนการค้นหาคำหลักของไฟล์ดรรชนีทั้งสองจะถูกจัดเรียงลำดับตามตัวอักษร จากนั้นเมื่อเริ่มกระบวนการค้นหา คำหลักทดสอบจะถูกนำมาใช้จากโหนดราก หากพบว่าโหนดรากว่างเปล่า กระบวนการนี้จะสิ้นสุดลงโดยไม่มีค่าส่งกลับ อย่างไรก็ตาม เมื่อโหนดรากมีคำหลักอยู่ จะมีการเปรียบเทียบคำหลักที่ต้องการกับคำหลักในโหนดนั้นๆ หากมีความตรงกัน ตำแหน่งของคำหลักจะถูกส่งกลับ และกระบวนการค้นหาจะดำเนินต่อไป ในกรณีที่คำหลักที่ค้นหามีค่าน้อยกว่าคำหลักของโหนดปัจจุบัน การค้นหาจะเปลี่ยนไปยังส่วนที่น้อยกว่าของต้นไม้ ซึ่งอยู่ทางด้านซ้าย ในทางตรงกันข้าม ถ้าคำหลักมีค่ามากกว่าคำหลักของโหนด การค้นหาจะดำเนินต่อในส่วนที่มากกว่าของต้นไม้ ซึ่งอยู่ทางด้านขวา การพบโหนดว่างในระหว่างกระบวนการค้นหาบ่งชี้ว่าคำหลักที่ต้องการไม่มีอยู่ภายในต้นไม้ค้นหาแบบทวิภาค จึงไม่มีค่าใดๆ ถูกส่งกลับ

อัลกอริทึม 3. ดรรชนีแบบหนาแน่นและดรรชนีแบบกระจาย โดยใช้เทคนิคเรียกว่า Binary Search Tree (BST)

Input: Dense and Sparse Index position File, keywords test with distribution as described in section 3.1
Output: Position in Bigdata.json

```
def binary_search_tree(index_file, keywords):
    bst = BinarySearchTree()
    for keyword, position in index_file:
        bst.insert(keyword, position)
    result = {}
    for desired_keyword in keywords:
        position = bst.search(desired_keyword)
        result[desired_keyword] = position
    return result
```

การใช้อัลกอริทึมดรรชนีแบบหนาแน่น และดรรชนีแบบกระจาย ร่วมกับเทคนิคต้นไม้ค้นหาแบบทวิภาค (Binary Search Tree, BST) เพื่อเข้าถึงตำแหน่งของคำหลักในไฟล์ดรรชนีทั้งสองประเภท โดยนำเข้าคำหลักทดสอบที่มีรูปแบบการแจกแจงที่แตกต่างกัน ได้แก่ การแจกแจงแบบปกติ แบบเอียงขวา และแบบเอียงซ้าย จากนั้นคำหลักเหล่านี้จะถูกนำเข้าสู่กระบวนการทำงานของต้นไม้ค้นหาแบบทวิภาค ในการประเมินประสิทธิภาพการเข้าถึงตำแหน่งของคำหลัก ในสถานการณ์เฉลี่ยหรือใช้เวลานานที่สุดคือ $O(n)$ โดยเฉพาะเมื่อตำแหน่งของคำหลักไม่ตรงกับตำแหน่งในไฟล์ดรรชนี ในทางตรงกันข้าม ในสถานการ์เวลาที่ดีที่สุดคือ $O(1)$ เกิดขึ้นเมื่อคำหลักที่ทดสอบตรงกับตำแหน่งในไฟล์ดรรชนีโดยตรง ซึ่งนำไปสู่การค้นพบตำแหน่งของไฟล์ Bigdata.json อย่างรวดเร็ว

3.5.4 อัลกอริทึมดรรชนีแบบหนาแน่นและดรรชนีแบบกระจาย โดยใช้เทคนิค Adelson–Velskii Landis Tree (AVL Tree)

สำหรับวิธีการค้นหาตำแหน่งของคำหลักภายในไฟล์ดรรชนีแบบหนาแน่น (Dense Index) และดรรชนีแบบกระจาย (Sparse Index) โดยใช้เทคนิค Adelson–Velskii Landis Tree (AVL Tree) แสดงดังอัลกอริทึม 4 นำเข้าคำหลักทดสอบที่มีการแจกแจงที่แตกต่างกันซึ่งประกอบด้วยการแจกแจงแบบปกติ แบบเอียงขวา และแบบเอียงซ้าย กระบวนการค้นหาคำหลักในไฟล์ดรรชนีด้วยวิธี AVL Tree เริ่มต้นด้วยการจัดเรียงข้อมูลตามลำดับตัวอักษรเพื่อสร้างโครงสร้างต้นไม้ที่สมดุลและมีประสิทธิภาพในการค้นหา โดยการเริ่มค้นหาจากโหนดรากของ AVL Tree การเปรียบเทียบคำหลักที่ต้องการค้นหากับคำหลักของโหนดปัจจุบันจะนำไปสู่

การตัดสินใจเลือกทิศทางการค้นหา เมื่อคำหลักที่ต้องการตรงกับคำหลักของโหนดปัจจุบัน ตำแหน่งของคำหลักนั้นจะถูกส่งกลับและกระบวนการค้นหาจะดำเนินการต่อไปยังคำหลักถัดไปที่ต้องการค้นหา หากคำหลักที่ต้องการมีค่าน้อยกว่าคำหลักของโหนดปัจจุบัน การค้นหาจะดำเนินการไปยังโหนดย่อยด้านซ้าย และในทางกลับกัน หากคำหลักที่ต้องการมีค่ามากกว่าการค้นหาจะเปลี่ยนไปยังโหนดย่อยด้านขวา กระบวนการนี้จะดำเนินต่อไปจนกว่าจะพบโหนดที่มีคำหลักตรงกับที่ค้นหา หรือจนกว่าจะไม่มีโหนดย่อยเหลืออยู่ในต้นไม้ ซึ่งจะหมายความว่าไม่พบคำหลักที่ต้องการในไฟล์ดรรชนี

อัลกอริทึม 4. ดรรชนีแบบหนาแน่นและดรรชนีแบบกระจาย โดยใช้เทคนิคเรียกว่า Adelson–Velskii Landis Tree search (AVL Tree)

Input: Dense and Sparse Index position File, keywords test with distribution as described in section 3.1
Output: Position in Bigdata.json

```
def avl_tree_search(index_file, keywords):
    avl = AVLTree()
    for keyword, position in index_file:
        avl.insert(keyword, position)
    result = {}
    for desired_keyword in keywords:
        position = avl.search(desired_keyword)
        result[desired_keyword] = position
    return result
```

การใช้อัลกอริทึมดรรชนีแบบหนาแน่น และดรรชนีแบบกระจาย ร่วมกับเทคนิค Adelson–Velskii Landis Tree (AVL Tree) เพื่อเข้าถึงตำแหน่งของคำหลักในไฟล์ดรรชนีทั้งสองประเภท โดยนำเข้าคำหลักทดสอบที่มีรูปแบบการแจกแจงที่แตกต่างกัน ได้แก่ การแจกแจงแบบปกติ แบบเอียงขวา และแบบเอียงซ้าย กระบวนการทดสอบคำหลักดำเนินการผ่านเทคนิค AVL Tree ซึ่งทำให้สามารถเข้าถึงตำแหน่งของคำหลักได้อย่างมีประสิทธิภาพ โดยมีความซับซ้อนของเวลาการประมวลผลคือ $O(log\ n)$ สำหรับทั้งสถานการณ์เฉลี่ยและกรณีที่เลวร้ายที่สุด หากคำหลักที่ทดสอบตรงกันกับไฟล์ดรรชนี 'Bigdata.json' ตำแหน่งของคำหลักนั้นจะถูกคืนค่ากลับอย่างรวดเร็ว

บทที่ 4

ผลการวิจัย

ในการวิจัยครั้งนี้ ผู้วิจัยประเมินประสิทธิภาพของเทคนิคการทำดรรชนีที่พัฒนาขึ้นใหม่สำหรับฐานข้อมูลไม่ใช่เชิงสัมพันธ์ (Not only Structured Query Language, NoSQL) จัดเก็บข้อมูลภายใต้ไฟล์รูปแบบวัตถุของจาวาสคริปต์ (JavaScript Object Notation, JSON) ของการสำรวจและเข้าถึงชุดข้อมูลขนาดใหญ่ โดยเฉพาะอย่างยิ่ง เมื่อเปรียบเทียบประสิทธิภาพของดรรชนีแบบหนาแน่น (Dense Index) และดรรชนีแบบกระจาย (Sparse Index) ทั้งที่มีและไม่มีดรรชนีผู้วิจัยทดลองบนระบบคอมพิวเตอร์ที่มีหน่วยประมวลผลกลาง (Central Processing Unit, CPU) ของ Intel™ ที่มีความเร็วสูงสุดถึง 4.90 กิกะเฮิร์ตซ์ พร้อมกับหน่วยความจำเข้าถึงแบบสุ่ม (Random Access Memory, RAM) ขนาด 8 กิกะไบต์ และโซลิดสเตตไดรฟ์ (Solid State Drive, SSD) ขนาด 500 กิกะไบต์ ผู้วิจัยนำเข้าชุดข้อมูลขนาดใหญ่ที่เรียกว่า Bigdata.json เพื่อให้ได้ตัวอย่างที่เป็นตัวแทน ผู้วิจัยสุ่มเลือกข้อมูล 524 รายการ จากฟิลด์ละติจูด (Latitude) ลองจิจูด (Longitude) และ ชื่อแหล่งที่มา (Resource_name) ซึ่งคิดเป็น 0.05% ของข้อมูลจำลองทั้งหมด และทดสอบอัลกอริทึมดังกล่าวซ้ำสามครั้ง การวิจัยนี้มีวัตถุประสงค์เพื่อตรวจสอบประสิทธิภาพเวลาในการสำรวจและเข้าถึงข้อมูล โดยมีและไม่มีดรรชนีรวมถึงมีข้อมูลหรือไม่มีข้อมูลที่ระบุภายในไฟล์ Bigdata.json โดยมีผลลัพธ์ของการทดลองดังนี้ ในการทดลองครั้งแรก ผู้วิจัยประเมินประสิทธิภาพของการเข้าถึงข้อมูลโดยการใช้ดรรชนีแบบหนาแน่น (Dense Index) ในการสำรวจและเข้าถึงข้อมูลจากไฟล์ Bigdata.json โดยเปรียบเทียบประสิทธิภาพทางด้านเวลาของการเข้าถึงข้อมูลแบบหนึ่งต่อหนึ่ง (One to one) ทั้งที่มีดรรชนีแบบหนาแน่น (Dense Index) และไม่มีดรรชนี (Non Index) เป็นที่น่าสังเกตว่าคำหลัก (Keyword) ที่เลือกแบบสุ่มทั้งหมดมีอยู่ในไฟล์ Bigdata.json การทดลองนี้มีวัตถุประสงค์เพื่อตรวจสอบประสิทธิภาพเวลาในการสำรวจและเข้าถึงข้อมูลของการใช้ดรรชนีแบบหนาแน่น (Dense Index) เปรียบเทียบกับการเข้าถึงข้อมูลโดยไม่มีดรรชนี (Non Index) โดยเน้นที่การมีอยู่ของคำหลักภายในไฟล์ Bigdata.json ดังแสดงในตาราง 2 ของผลการทดสอบที่เกี่ยวข้องกับการเรียกคำหลัก 524 คำ โดยใช้ดรรชนีแบบหนาแน่น (Dense Index) ดำเนินการวิจัยทั้งหมดสามรอบ โดยนำเสนอเวลาเฉลี่ยในการเข้าถึงข้อมูลต่อคำหลัก (Keyword) การค้นพบนี้แสดงให้เห็นว่าการใช้ดรรชนีแบบหนาแน่น (Dense Index) สามารถช่วยลดเวลาในการเข้าถึงข้อมูลเมื่อ

เทียบกับไม่มีดรรชนี (Non Index) ผลการทดสอบพบว่า ดรรชนีแบบหนาแน่น (Dense Index) ใช้เวลาเฉลี่ยต่อคำหลักในการเข้าถึงตำแหน่งข้อมูลเท่ากับ 375.721 มิลลิวินาที ส่วนการเข้าถึงไฟล์ Bigdata.json โดยการแบ่งส่วนข้อมูล (Segment) ใช้เวลาเฉลี่ยต่อคำหลักเท่ากับ 6.476 มิลลิวินาที นอกจากนี้ เมื่อพิจารณาการเข้าถึงข้อมูลทั้งหมดด้วยการใช้ดรรชนีแบบหนาแน่น (Dense Index) เวลาที่ใช้เฉลี่ยต่อคำหลักเท่ากับ 382.196 มิลลิวินาที หากไม่มีดรรชนี (Non Index) ใช้เวลาในการเข้าถึงข้อมูลเฉลี่ยต่อคำหลักเท่ากับ 26,869.218 มิลลิวินาที ในการทดสอบซ้ำทั้งสามรอบของ 524 คำหลัก พบว่าการเข้าถึงข้อมูลโดยใช้ดรรชนีแบบหนาแน่น (Dense Index) ใช้เวลาเฉลี่ยเท่ากับ 200,270.949 มิลลิวินาที ในขณะที่การเข้าถึงข้อมูลโดยไม่มีดรรชนี (Non Index) ใช้เวลาโดยเฉลี่ยเท่ากับ 14,079,470.37 มิลลิวินาที ผลการทดสอบแสดงให้เห็นถึงประสิทธิภาพทางด้านเวลาของ ดรรชนีแบบหนาแน่น (Dense Index) ที่ดีขึ้นเป็นอย่างมาก เมื่อเทียบกับไม่มีดรรชนี (Non Index) เนื่องจากสามารถข้ามข้อมูลที่ไม่จำเป็นและเข้าถึงข้อมูลเพียงครั้งเดียว นอกจากนี้ ยังนำเข้าข้อมูลลงในหน่วยความจำ หนึ่งครั้งต่อคำหลัก (Keyword) ช่วยลดเวลาในการเข้าถึงข้อมูลและประมวลผลข้อมูล ในทางตรงกันข้าม การเข้าถึงข้อมูลโดยไม่มีดรรชนี (Non Index) จะนำเข้าข้อมูลคำหลัก (Keyword) ทั้งหมด ซึ่งนำไปสู่การสูญเสียทรัพยากรหน่วยความจำอย่างมากและส่งผลกระทบต่อเวลาในการเข้าถึงข้อมูลที่เพิ่มขึ้นตามขนาดข้อมูล

ตาราง 1 การเปรียบเทียบประสิทธิภาพของการสำรวจและเวลาในการเข้าถึงไฟล์ข้อมูล Bigdata.json โดยใช้ดรรชนีแบบหนาแน่น และไม่มีการจัดทำดรรชนี

การประเมินประสิทธิภาพ	**524 คำหลัก รอบที่ 1 (มิลลิวินาที)**	**524 คำหลัก รอบที่ 2 (มิลลิวินาที)**	**524 คำหลัก รอบที่ 3 (มิลลิวินาที)**	**ค่าเฉลี่ย 524 คำหลัก (มิลลิวินาที)**	**ค่าเฉลี่ย 1 คำหลัก (มิลลิวินาที)**
เวลาเข้าถึงข้อมูล Index ด้วย Dense Index	197,367.708	197,722.472	195,542.527	196,877.569	375.721
เวลาเข้าถึงข้อมูล Segment ด้วย Dense index	3,388.343	3,388.343	3,388.343	3,393.38	6.476
เวลาเข้าถึงข้อมูล Dense index ทั้งหมด	200,756.5	201,112.118	198,944.677	200,270.949	382.196
เวลาการเข้าถึงข้อมูล กรณีไม่มีดรรชนี	14,091,572.26	14,087,301.7	14,059,537.15	14,079,470.37	26,869.218

การทดลองครั้งที่สอง ผู้วิจัยประเมินประสิทธิภาพของการเข้าถึงข้อมูลแบบใช้ดรรชนีแบบหนาแน่น (Dense Index) ในการสำรวจและเข้าถึงข้อมูลจากไฟล์ Bigdata.json โดยเปรียบเทียบประสิทธิภาพทางด้านเวลาของการเข้าถึงข้อมูลแบบหนึ่งต่อหนึ่ง (One to one)ทั้งที่มีดรรชนีแบบหนาแน่น (Dense Index และประสิทธิภาพของการเข้าถึงข้อมูลแบบไม่มีดรรชนี (Non Index) โดยไฟล์ดรรชนีดังกล่าวจะไม่พบคำหลัก (Keyword not found) อันเนื่องมาจากใช้เทคนิคการเลือกแบบสุ่มทั้งหมด การวิจัยให้ผลลัพธ์ของการทดลองดังแสดงในตาราง 3 ของสถานการณ์ที่ไม่พบคำหลัก (Keyword not found) เวลาในการสำรวจและเข้าถึงข้อมูล ซึ่งจำกัดอยู่ที่เวลาที่ใช้ในการเข้าถึงข้อมูลในไฟล์ดรรชนีแบบหนาแน่น (Dense Index) และไม่เกี่ยวข้องกับการเข้าถึงข้อมูลในหน่วยความจำ นี่เป็นเพราะไม่มีส่วนที่เกี่ยวข้องที่จะนำเข้าและไม่มีตำแหน่ง (Position) ในดรรชนีแบบหนาแน่น (Dense Index) ที่ต้องการเข้าถึงข้อมูล ผู้วิจัยทำการทดลองโดยใช้คำหลักของดรรชนีแบบหนาแน่น (Dense Index) 524 คำหลัก ผลลัพธ์แสดงให้เห็นถึงประสิทธิภาพทางด้านเวลาของการเข้าถึงข้อมูลโดยเฉลี่ยต่อคำหลักเท่ากับ 1.097 มิลลิวินาที โดยมีเวลาเข้าถึงข้อมูลเฉลี่ยสำหรับทั้งสามรอบเท่ากับ 574.767 มิลลิวินาที ในทางตรงกันข้าม วิธีการเข้าถึงข้อมูลโดยไม่มีดรรชนี (Non Index) ในกรณีที่ไม่พบนั้นทดสอบกับ 524 คำหลัก ผลลัพธ์พบว่าเวลาในการเข้าถึงข้อมูลโดยเฉลี่ยต่อคำหลักเท่ากับ 38,300.848 มิลลิวินาที โดยมีเวลาการเข้าถึงข้อมูลเฉลี่ยสำหรับทั้งสามรอบเท่ากับ 20,069,244.44 มิลลิวินาที ผลลัพธ์พบว่าการเข้าถึงข้อมูลโดยดรรชนีแบบหนาแน่น (Dense Index) ในกรณีที่ไม่พบคำหลัก (Keyword not found) มีประสิทธิภาพมากกว่า กรณีที่ไม่มีดรรชนี (Non Index) เนื่องจากข้อมูลที่ไม่จำเป็นถูกข้ามไป และข้อมูลถูกเข้าถึงเพียงครั้งเดียวต่อคำหลัก (Keyword) ทำให้ลดเวลาในการเข้าถึงข้อมูลเฉลี่ยต่อคำหลัก (Keyword) ลดลงเหลือเพียง 1.097 มิลลิวินาที ในทางกลับกัน การสำรวจและเข้าถึงข้อมูลโดยไม่มีการจัดทำดรรชนีในกรณีที่ไม่พบคำหลัก (Keyword not found) ทำให้ใช้เวลาในการสำรวจและเข้าถึงข้อมูลนานกว่ามาก โดยเวลาเข้าถึงข้อมูลเฉลี่ยต่อคำหลักเท่ากับ 38,300.848 มิลลิวินาที ส่งผลกระทบสูงต่อหน่วยความจำ โดยเฉพาะอย่างยิ่งในกระบวนการนำเข้าคำหลักทั้งหมด และเวลาในการเข้าถึงของข้อมูล

ตาราง 2 การเปรียบเทียบประสิทธิภาพของการเข้าถึงข้อมูล โดยใช้ดรรชนีแบบหนาแน่นและไม่มีดรรชนีและกรณีที่ไม่มีคำหลักในไฟล์ข้อมูล Bigdata.json

การประเมินประสิทธิภาพ	524 คำหลัก รอบที่ 1 (มิลลิวินาที)	524 คำหลัก รอบที่ 2 (มิลลิวินาที)	524 คำหลัก รอบที่ 3 (มิลลิวินาที)	ค่าเฉลี่ย 524 คำหลัก (มิลลิวินาที)	ค่าเฉลี่ย 1 คำหลัก (มิลลิวินาที)
การเข้าถึงข้อมูลด้วย Dense Index ในกรณีที่ไม่พบคำหลัก	569.96	572.78	581.56	574.767	1.097
การเข้าถึงข้อมูลด้วย Non Index ในกรณีที่ไม่พบคำหลัก	20,528,300.2	20,871,312.76	1,880,8120.53	20,069,244.44	38,300.848

ในการทดลองครั้งที่สาม ผู้วิจัยประเมินประสิทธิภาพการเข้าถึงข้อมูลของการใช้ดรรชนีแบบกระจาย (Sparse Index) ในการสำรวจและเข้าถึงข้อมูลจากไฟล์ Bigdata.json โดยเปรียบเทียบประสิทธิภาพทางด้านเวลาของการเข้าถึงข้อมูลแบบหนึ่งต่อกลุ่ม (One-to-Many) ทั้งที่มีดรรชนีแบบกระจาย (Sparse Index) และไม่มีดรรชนี (Non Index) เป็นที่น่าสังเกตว่าคำหลัก (Keyword) ที่เลือกแบบสุ่มทั้งหมดมีอยู่ในไฟล์ Bigdata.json การทดลองนี้มีวัตถุประสงค์เพื่อตรวจสอบประสิทธิภาพเวลาในการสำรวจและเข้าถึงข้อมูลของการใช้ดรรชนีแบบกระจาย (Sparse Index) เปรียบเทียบกับไม่มีดรรชนี (Non Index)โดยเน้นที่การมีอยู่ของคำหลักภายในไฟล์ Bigdata.json ดังแสดงในตาราง 4 ของผลการทดสอบที่เกี่ยวข้องกับการเรียกคำหลัก 524 คำ โดยใช้ดรรชนีแบบกระจาย (Sparse Index) ดำเนินการวิจัยทั้งหมดสามรอบ โดยนำเสนอเวลาเฉลี่ยในการเข้าถึงข้อมูลต่อคำหลัก (Keyword) การค้นพบนี้แสดงให้เห็นว่าการใช้ดรรชนีแบบกระจาย (Sparse Index) สามารถช่วยลดเวลาในการเข้าถึงข้อมูลเมื่อเทียบกับไม่มีดรรชนี (Non Index)ผลการทดสอบพบว่า ดรรชนีแบบกระจาย (Sparse Index) ใช้เวลาเฉลี่ยต่อคำหลักในการเข้าถึงหลายตำแหน่งข้อมูลเท่ากับ 9.795มิลลิวินาที ส่วนการเข้าถึงไฟล์ Bigdata.json โดยการแบ่งส่วนข้อมูล (Segment) ใช้เวลาเฉลี่ยต่อคำหลักเท่ากับ 756.65 มิลลิวินาที นอกจากนี้ เมื่อพิจารณาการเข้าถึงข้อมูลทั้งหมดด้วยการใช้ดรรชนีแบบกระจาย (Sparse Index) เวลาที่ใช้เฉลี่ยต่อคำหลักเท่ากับ 854.662 มิลลิวินาที หากไม่มีดรรชนี (Non Index)ใช้เวลาในการเข้าถึงข้อมูลเฉลี่ยต่อคำหลักเท่ากับ 55,197.734 มิลลิวินาที ในการทดสอบซ้ำทั้งสามรอบของ 524 คำหลัก พบว่าการเข้าถึงข้อมูลโดยใช้ดรรชนีแบบกระจาย (Sparse Index) ใช้เวลาเฉลี่ยเท่ากับ 447,842.638 มิลลิวินาที ในขณะที่การเข้าถึงข้อมูลโดยไม่มีดรรชนี (Non Index)ใช้เวลาโดยเฉลี่ยเท่ากับ 28,923,612.59 มิลลิวินาที ผลการทดสอบแสดงให้เห็นถึงประสิทธิภาพทางด้านเวลาของ ดรรชนีแบบกระจาย (Sparse Index) ที่ดีขึ้นเป็นอย่างมาก เมื่อเทียบกับไม่มีดรรชนี (Non Index) เนื่องจากสามารถข้ามข้อมูลที่ไม่จำเป็นและ

เข้าถึงข้อมูลเพียงครั้งเดียว นอกจากนี้ ยังนำเข้าข้อมูลลงในหน่วยความจำหนึ่งครั้งต่อคำหลัก (Keyword) แม้ว่าคำหลักจะปรากฏในหลายตำแหน่ง (Position) ซึ่งสามารถช่วยลดเวลาในการเข้าถึงข้อมูลและประมวลผลข้อมูล ในทางตรงกันข้าม การเข้าถึงข้อมูลโดยไม่มีดรรชนี (Non Index)จะนำเข้าข้อมูลคำหลัก (Keyword) ทั้งหมด ซึ่งนำไปสู่การสูญเสียทรัพยากรหน่วยความจำอย่างมากและส่งผลกระทบต่อเวลาในการเข้าถึงข้อมูลที่เพิ่มขึ้นตามขนาดข้อมูล

ตาราง 3 การเปรียบเทียบประสิทธิภาพของการสำรวจและเวลาในการเข้าถึงไฟล์ข้อมูล Bigdata.json โดยใช้ดรรชนีแบบกระจาย และไม่มีการจัดทำดรรชนี

ประเภทของการทดสอบ	524 คำหลัก รอบที่ 1 (มิลลิวินาที)	524 คำหลัก รอบที่ 2 (มิลลิวินาที)	524 คำหลัก รอบที่ 3 (มิลลิวินาที)	ค่าเฉลี่ย 524 คำหลัก (มิลลิวินาที)	ค่าเฉลี่ย 1 คำหลัก (มิลลิวินาที)
เวลาเข้าถึงข้อมูล Index ด้วย Sparse Index	52,056.211	50,885.636	51,130.898	51,357.582	9.795
เวลาเข้าถึง Segment ด้วย Sparse index	395,083.445	395,693.442	398,678.282	396,485.565	756.65
เวลาเข้าถึงข้อมูล Sparse Index ทั้งหมด	447,139.656	446,579.078	449,809.18	447,842.638	854.661
เวลาเข้าถึงข้อมูล กรณีไม่มีดรรชนี	28,850,300.91	28,970,251.46	28,950,285.38	28,923,612.59	55,197.734

การทดลองครั้งที่สี่ ผู้วิจัยประเมินประสิทธิภาพการเข้าถึงข้อมูลของการใช้ดรรชนีแบบกระจาย (Sparse Index) ในการสำรวจและเข้าถึงข้อมูลจากไฟล์ Bigdata.json โดยเปรียบเทียบประสิทธิภาพทางด้านเวลาของการเข้าถึงข้อมูลแบบหนึ่งต่อกลุ่ม (One-to-Many) ทั้งที่มีดรรชนีแบบกระจาย (Sparse Index) และไม่มีดรรชนี (Non Index) โดยไฟล์ดรรชนีดังกล่าวจะไม่พบคำหลัก (Keyword not found) การวิจัยให้ผลลัพธ์ของการทดลองดังแสดงในตาราง 5 ของสถานการณ์ที่ไม่พบคำหลัก (Keyword not found) เวลาในการสำรวจและเข้าถึงข้อมูล ซึ่งจำกัดอยู่ที่เวลาที่ใช้ในการเข้าถึงข้อมูลในไฟล์ดรรชนีแบบกระจาย (Sparse Index) และไม่เกี่ยวข้องกับการเข้าถึงข้อมูลในหน่วยความจำเข้าถึงแบบสุ่ม (Random Access Memory, RAM) นี่เป็นเพราะไม่มีส่วนที่เกี่ยวข้องที่จะนำเข้าและไม่มีตำแหน่ง (Position) ในดรรชนีแบบกระจาย (Sparse Index) ที่ต้องการเข้าถึงข้อมูล ผู้วิจัยทำการทดลองโดยใช้คำหลักของดรรชนีแบบกระจาย (Sparse Index) 524 คำหลัก ผลลัพธ์แสดงให้เห็นถึงประสิทธิภาพทางด้านเวลาของ

การเข้าถึงข้อมูลโดยเฉลี่ยต่อคำหลักเท่ากับ 0.179 มิลลิวินาที โดยมีเวลาเข้าถึงข้อมูลเฉลี่ยสำหรับทั้งสามรอบเท่ากับ 93.543 มิลลิวินาที ในทางตรงกันข้าม วิธีการเข้าถึงข้อมูลโดยไม่มีดรรชนี (Non Index)ในกรณีที่ไม่พบคำหลักนั้นทดสอบกับ 524 คำหลัก ผลลัพธ์พบว่าเวลาในการเข้าถึงข้อมูลโดยเฉลี่ยต่อคำหลักเท่ากับ 47,203.253 มิลลิวินาที โดยมีเวลาการเข้าถึงข้อมูลเฉลี่ยสำหรับทั้งสามรอบเท่ากับ 24,734,504.72 มิลลิวินาที ผลลัพธ์พบว่าการเข้าถึงข้อมูลโดยดรรชนีแบบกระจาย (Sparse Index) ในกรณีที่ไม่พบคำหลัก (Keyword not found) มีประสิทธิภาพมากกว่า กรณีที่ไม่มีดรรชนี (Non Index) เนื่องจากข้อมูลที่ไม่จำเป็นถูกข้ามไปและข้อมูลถูกเข้าถึงเพียงครั้งเดียวต่อคำหลัก (Keyword) ทำให้ลดเวลาในการเข้าถึงข้อมูลเฉลี่ยต่อคำหลัก (Keyword) ลดลงเหลือเพียง 0.179 มิลลิวินาที ในทางกลับกัน การสำรวจและเข้าถึงข้อมูลโดยไม่มีการจัดทำดรรชนีในกรณีที่ไม่พบคำหลัก (Keyword not found) ทำให้ใช้เวลาในการสำรวจและเข้าถึงข้อมูลนานกว่ามาก โดยเวลาเข้าถึงข้อมูลเฉลี่ยต่อคำหลักเท่ากับ 47,203.253 มิลลิวินาที ส่งผลกระทบสูงต่อหน่วยความจำ โดยเฉพาะอย่างยิ่งเวลาในกระบวนการนำเข้าคำหลักทั้งหมด และเวลาในการเข้าถึงข้อมูล

ตาราง 4 การเปรียบเทียบประสิทธิภาพของการเข้าถึงข้อมูล โดยใช้ดรรชนีแบบกระจายและไม่มีดรรชนีและกรณีที่ไม่มีคำหลักในไฟล์ข้อมูล Bigdata.json

การประเมินประสิทธิภาพ	**524 คำหลัก รอบที่ 1 (มิลลิวินาที)**	**524 คำหลัก รอบที่ 2 (มิลลิวินาที)**	**524 คำหลัก รอบที่ 3 (มิลลิวินาที)**	**ค่าเฉลี่ย 524 คำหลัก (มิลลิวินาที)**	**ค่าเฉลี่ย 1 คำหลัก (มิลลิวินาที)**
การเข้าถึงข้อมูลด้วย Sparse Index ในกรณีที่ไม่พบคำหลัก	93.12	90.2	97.31	93.543	0.179
การเข้าถึงข้อมูลด้วย No Index ในกรณีที่ไม่พบคำหลัก	26,873,520.89	23,599,661.86	23,730,331.4	24,734,504.72	47,203.253

วัตถุประสงค์หนึ่งของการทดลองนี้คือการกำหนดขนาดส่วนที่เหมาะสมที่สุด โดยการทดสอบในสามช่วงที่แตกต่างกัน ช่วงแรกครอบคลุมตั้งแต่ 100 ถึง 1,000 โดยเพิ่มขึ้นทีละ 100 สำหรับแต่ละขนาดส่วน ช่วงที่สองครอบคลุม 1,000 ถึง 10,000 โดยเพิ่มขึ้น 1,000 สำหรับแต่ละขนาดส่วน และช่วงสุดท้ายขยายจาก 10,000 เป็น 200,000 โดยเพิ่มขึ้น 10,000 สำหรับแต่ละขนาดส่วน ผู้วิจัยได้ทำการทดลองทั้งหมดสามรอบและคำนวณค่าเฉลี่ย นอกจากนี้ ตารางที่ 6 และ

7 ยังสรุปค่าขนาดส่วนที่เหมาะสมสำหรับการประเมินประสิทธิภาพของดรรชนีแบบหนาแน่นและดรรชนีแบบกระจายในช่วงที่กำหนดทั้งสามช่วง โดยมีรายละเอียดของผลการทดลองดังนี้

ตาราง 6 แสดงค่าขนาดส่วน (Segment size) ที่เหมาะสมสำหรับการประเมินประสิทธิภาพของดรรชนีแบบหนาแน่น (Dense Index) โดยแบ่งออกเป็นสามช่วง แต่ละช่วงขนาดส่วน (Segment size) กลุ่มแรกตั้งแต่ 100 ถึง 1,000 มีเวลาเฉลี่ยต่อคำหลักเท่ากับ 197,367.708 มิลลิวินาที ช่วงที่สองตั้งแต่ 1,000 ถึง 10,000 มีเวลาเฉลี่ยต่อคำหลักเท่ากับ 197722.472 มิลลิวินาที สุดท้าย ช่วงที่สามตั้งแต่ 10,000 ถึง 200,000 มีเวลาเฉลี่ยต่อคำหลักเท่ากับ 195542.527 มิลลิวินาที ภาพ 33 แสดงให้เห็นว่าขนาดส่วน (Segment size) ในช่วงตั้งแต่ 100 ถึง 200,000 พบว่าประสิทธิภาพในการเข้าถึงคำหลัก (Keyword) มีความแตกต่างทางด้านของเวลาเล็กน้อย เนื่องจากการเข้าถึงหน่วยความจำ ไม่มีความแตกต่าง เมื่อเทียบกับขนาดส่วน (Segment size) ทั้งนี้ขึ้นอยู่กับประสิทธิภาพของระบบคอมพิวเตอร์ ดังนั้น ไม่ว่าขนาดส่วน (Segment size) จะอยู่ในช่วงใด ดรรชนีแบบหนาแน่น (Dese Index) ยังคงสามารถจัดการกับหน่วยความจำและการเข้าถึงคำหลัก (Keyword) ได้อย่างรวดเร็ว

ตาราง 5 ประสิทธิภาพดรรชนีแบบหนาแน่น ตามขนาดส่วน และความถี่ของคำหลัก

ขนาดของส่วน	**ค่าเฉลี่ย 1 คำสำคัญ (มิลลิวินาที)**
100 to 1000	197,367.708
1000 to 10000	197722.472
10000 to 200000	195542.527

ภาพ 31 การเปรียบเทียบประสิทธิภาพของดรรชนีแบบหนาแน่น

ตาราง 7 แสดงค่าขนาดส่วน (Segment size) ที่เหมาะสมสำหรับการประเมินประสิทธิภาพของดรรชนีแบบกระจาย. (Sparse Index) โดยแบ่งออกเป็นสามช่วง แต่ละช่วงขนาดส่วน (Segment size) กลุ่มแรกตั้งแต่ 100 ถึง 1,000 มีเวลาเฉลี่ยต่อคำหลักเท่ากับ 52,056.21 มิลลิวินาที ช่วงที่สองตั้งแต่ 1,000 ถึง 10,000 มีเวลาเฉลี่ยต่อคำหลักเท่ากับ 50,885.637 มิลลิวินาที สุดท้าย ช่วงที่สามตั้งแต่ 10,000 ถึง 200,000 มีเวลาเฉลี่ยต่อคำหลักเท่ากับ 51,130.898 มิลลิวินาที ภาพ 34 แสดงให้เห็นว่าขนาดส่วน (Segment size) ในช่วงตั้งแต่ 100 ถึง 200,000 พบว่าประสิทธิภาพในการเข้าถึงคำหลัก (Keyword) มีความแตกต่างทางด้านของเวลาเล็กน้อย เนื่องจากการเข้าถึงหน่วยความจำ ไม่มีความแตกต่าง เมื่อเทียบกับขนาดส่วน (Segment size) ทั้งนี้ขึ้นอยู่กับประสิทธิภาพของระบบคอมพิวเตอร์ ดังนั้น ไม่ว่าขนาดส่วน (Segment size) จะอยู่ในช่วงใด ดรรชนีแบบกระจาย (Sparse Index) ยังคงสามารถจัดการกับหน่วยความจำและการเข้าถึงคำหลัก (Keyword) ได้อย่างรวดเร็ว

ตาราง 6 ประสิทธิภาพดรรชนีแบบหนาแน่น ตามขนาดส่วน และความถี่ของคำหลัก

ขนาดของส่วน	**ค่าเฉลี่ย 1 คำสำคัญ (มิลลิวินาที)**
100 to 1000	52,056.21
1000 to 10000	50,885.637
10000 to 200000	51,130.898

***ภาพ* 32** การเปรียบเทียบประสิทธิภาพของดรรชนีแบบกระจาย

โปรดสังเกตว่า การสำรวจและเข้าถึงข้อมูลโดยใช้ดรรชนีแบบหนาแน่น อาจใช้เวลานานกว่าดรรชนีแบบกระจาย สาเหตุมาจากการค้นหาคำหลัก ที่ต้องใช้เวลาดึงตำแหน่งข้อมูลในไฟล์ดรรชนีแบบหนาแน่นมากกว่า ความแตกต่างนี้เป็นไปตามขนาดไฟล์และจำนวนข้อมูล ไฟล์ดรรชนีแบบหนาแน่นมีขนาด 20 เมกะไบต์และมีข้อมูล 1,000,000 รายการ ในขณะที่ไฟล์ดรรชนีแบบกระจายมีขนาดเพียง 8 เมกะไบต์และมีข้อมูล 5146 รายการเท่านั้น

ในส่วนถัดไปนี้ เป็นการประเมินประสิทธิภาพเวลาของการเข้าถึงตำแหน่งข้อมูลจากไฟล์ Dense_index_position.json และ Sparse_index_position.json ด้วยเทคนิคการค้นหาเชิงเส้น (Linear search, LS) การค้นหาแบบทวิภาค (Binary search, BS) ต้นไม้ค้นหาแบบทวิภาค (Binary search Tree, BST) และ Adelson-Velskii Landis Tree (AVL Tree) การเปรียบเทียบนี้ได้ดำเนินการกับการแจกแจงคำหลักที่มีความแตกต่างกันทั้งสามรูปแบบ ได้แก่ การแจกแจงปกติ (Normal Distribution) การแจกแจงที่มีลักษณะเอียงทางซ้าย (Left-skewed Distribution) และการแจกแจงที่มีลักษณะเอียงทางขวา (Right-Skewed Distribution) ตามที่อธิบายไว้ในส่วน 3.4 ผู้วิจัยได้ดำเนินการทดลองแบ่งออกเป็นสองประเภท ได้แก่ ดรรชนีแบบหนาแน่น (Dense Index) และดรรชนีแบบกระจาย (Sparse Index) ดังแสดงในตารางที่ 8 ดรรชนีทั้งสองได้รับการทดสอบแบ่งออกเป็นสองลักษณะ ได้แก่ ลักษณะแรกคือเวลาเฉลี่ยทั้งหมด 15 รอบ ซึ่งเปรียบเทียบประสิทธิภาพของเวลาในการเข้าถึงตำแหน่งต่อคำหลัก 524 คำ และลักษณะที่สองคือค่าเฉลี่ยต่อคำหลัก ซึ่งเปรียบเทียบประสิทธิภาพของเวลาในการเข้าถึงตำแหน่งต่อคำหลัก

สำหรับการทดสอบเพื่อเปรียบเทียบประสิทธิภาพเวลาของการเข้าถึงตำแหน่งของดรรชนีแบบหนาแน่น และดรรชนีแบบกระจาย การเรียงลำดับดรรชนีทั้งสอง สามารถปรับปรุงประสิทธิภาพเวลาในการเข้าถึงตำแหน่งได้ โดยใช้เทคนิคที่เรียกว่าการค้นหาเชิงเส้น (Linear search, LS) การค้นหาแบบทวิภาค (Binary search, BS) ต้นไม้ค้นหาแบบทวิภาค (Binary search Tree, BST) และ Adelson-Velskii Landis Tree (AVL Tree) อย่างมีประสิทธิภาพ แสดงดังตาราง 8 โปรดทราบว่าเทคนิค "การค้นหาเชิงเส้น" ไม่จำเป็นต้องเรียงลำดับไฟล์ดรรชนีก่อนค้นหา นี่เป็นเพราะการค้นหาจะพิจารณารายการ ทั้งหมดในชุดข้อมูลโดยไม่คำนึงถึงลำดับของคำหลัก ในทางตรงกันข้ามการค้นหาแบบทวิภาค (BS) ต้นไม้ค้นหาแบบทวิภาค (BST) และ AVL Tree ต้องการการสร้างและการเรียงลำดับคำหลักดรรชนีแบบหนาแน่นและดรรชนีแบบกระจาย ผลการวิจัยพบว่าดรรชนีแบบหนาแน่นของเทคนิค "การค้นหาแบบทวิภาค (BS)" ใช้

เวลาเฉลี่ย 1795.708 มิลลิวินาที เทคนิค "ต้นไม้ค้นหาแบบทวิภาค (BST)" ใช้เวลาเฉลี่ย 773.149 มิลลิวินาที และเทคนิค AVL Tree ใช้เวลาเฉลี่ย 635.262 มิลลิวินาที ในขณะที่ดรรชนีแบบกระจาย เทคนิค "การค้นหาแบบทวิภาค (BS)" ใช้เวลาเฉลี่ย 136.711 มิลลิวินาที เทคนิค "ต้นไม้ค้นหาแบบทวิภาค (BST)" ใช้เวลาเฉลี่ย 113.538 มิลลิวินาทีและเทคนิค AVL Tree ใช้เวลาเฉลี่ย 137.335 มิลลิวินาที

ตาราง 7 การเปรียบเทียบประสิทธิภาพเวลาของการเข้าถึงตำแหน่งข้อมูลของดรรชนีแบบหนาแน่นและดรรชนีแบบกระจาย โดยใช้เทคนิคที่เรียกว่า LS, BS, BST และ AVL กับลักษณะคำหลักที่มีการแจกแจงที่แตกต่างกันทั้งสามรูปแบบดังนี้

ดรรชนีแบบหนาแน่น (Dense Index)	**การค้นหาเชิงเส้น (Linear search, LS)**		
	ค่าเฉลี่ย 15 รอบ (มิลลิวินาที)	**ค่าเฉลี่ยต่อคำหลัก (มิลลิวินาที)**	**เวลาเรียงลำดับไฟล์ดรรชนี (มิลลิวินาที)**
การดึงข้อมูลตามดรรชนีแบบหนาแน่นในการแจกแจงแบบปกติ	342259.155	653.166	ไม่มีการเรียงลำดับไฟล์ดรรชนี
การดึงข้อมูลตามดรรชนีแบบหนาแน่นในการแจกแจงแบบเอียงซ้าย	337220.625	643.551	
การดึงข้อมูลตามดรรชนีแบบหนาแน่นในการแจกแจงแบบเอียงขวา	349457.13	666.903	
ดรรชนีแบบหนาแน่น (Dense Index)	**การค้นหาแบบทวิภาค (Binary search, BS)**		
	ค่าเฉลี่ย 15 รอบ (มิลลิวินาที)	**ค่าเฉลี่ยต่อคำหลัก (มิลลิวินาที)**	**เวลาเรียงลำดับไฟล์ดรรชนี (มิลลิวินาที)**
การดึงข้อมูลตามดรรชนีแบบหนาแน่นในการแจกแจงแบบแบบปกติ	9.811	0.019	1795.708
การดึงข้อมูลตามดรรชนีแบบหนาแน่นในการแจกแจงแบบเอียงซ้าย	10.199	0.019	
การดึงข้อมูลตามดรรชนีแบบหนาแน่นในการแจกแจงแบบเอียงขวา	11.572	0.022	
ดรรชนีแบบหนาแน่น (Dense Index)	**ต้นไม้ค้นหาแบบทวิภาค (Binary search Tree, BST)**		
	ค่าเฉลี่ย 15 รอบ (มิลลิวินาที)	**ค่าเฉลี่ยต่อคำหลัก (มิลลิวินาที)**	**เวลาเรียงลำดับไฟล์ดรรชนี (มิลลิวินาที)**
การดึงข้อมูลตามดรรชนีแบบหนาแน่นในการแจกแจงแบบปกติ	179.970	0.343	773.149
การดึงข้อมูลตามดรรชนีแบบหนาแน่นในการแจกแจงแบบเอียงซ้าย	173.583	0.331	
การดึงข้อมูลตามดรรชนีแบบหนาแน่นในการแจกแจงแบบเอียงขวา	155.317	0.296	
ดรรชนีแบบหนาแน่น	Adelson–Velskii and Landis Tree (AVL Tree)		

(Dense Index)	ค่าเฉลี่ย 15 รอบ (มิลลิวินาที)	ค่าเฉลี่ยต่อคำหลัก (มิลลิวินาที)	เวลาเรียงลำดับไฟล์ดรรชนี (มิลลิวินาที)
การดึงข้อมูลตามดรรชนีแบบหนาแน่นในการแจกแจงแบบปกติ	2.703	0.005	635.262
การดึงข้อมูลตามดรรชนีแบบหนาแน่นในการแจกแจงแบบเอียงซ้าย	2.346	0.004	
การดึงข้อมูลตามดรรชนีแบบหนาแน่นในการแจกแจงแบบเอียงขวา	2.433	0.005	
ดรรชนีแบบกระจาย (Sparse Index)	**การค้นหาเชิงเส้น (Linear search, LS)**		
	ค่าเฉลี่ย 15 รอบ (มิลลิวินาที)	**ค่าเฉลี่ยต่อคำหลัก (มิลลิวินาที)**	**เวลาเรียงลำดับไฟล์ดรรชนี (มิลลิวินาที)**
การดึงข้อมูลตามดรรชนีแบบกระจายในการแจกแจงแบบปกติ	528112.194	1007.848	ไม่มีการเรียงลำดับไฟล์ดรรชนี
การดึงข้อมูลตามดรรชนีแบบกระจายในการแจกแจงแบบเอียงซ้าย	528253.244	1008.117	
การดึงข้อมูลตามดรรชนีแบบกระจายในการแจกแจงแบบเอียงขวา	528179.767	1007.977	
ดรรชนีแบบกระจาย (Sparse Index)	**การค้นหาแบบทวิภาค (Binary search, BS)**		
	ค่าเฉลี่ย 15 รอบ (มิลลิวินาที)	**ค่าเฉลี่ยต่อคำหลัก (มิลลิวินาที)**	**เวลาเรียงลำดับไฟล์ดรรชนี (มิลลิวินาที)**
การดึงข้อมูลตามดรรชนีแบบกระจายในการแจกแจงแบบปกติ	5.647	0.011	136.711
การดึงข้อมูลตามดรรชนีแบบกระจายในการแจกแจงแบบเอียงซ้าย	3.868	0.007	
การดึงข้อมูลตามดรรชนีแบบกระจายในการแจกแจงแบบเอียงขวา	5.592	0.011	
ดรรชนีแบบกระจาย (Sparse Index)	**ต้นไม้ค้นหาแบบทวิภาค (Binary search Tree, BST)**		
	ค่าเฉลี่ย 15 รอบ (มิลลิวินาที)	**ค่าเฉลี่ยต่อคำหลัก (มิลลิวินาที)**	**เวลาเรียงลำดับไฟล์ดรรชนี (มิลลิวินาที)**
การดึงข้อมูลตามดรรชนีแบบกระจายในการแจกแจงแบบปกติ	161.007	0.307	113.538
การดึงข้อมูลตามดรรชนีแบบกระจายในการแจกแจงแบบเอียงซ้าย	166.118	0.317	
การดึงข้อมูลตามดรรชนีแบบกระจายในการแจกแจงแบบเอียงขวา	158.378	0.302	
ดรรชนีแบบกระจาย (Sparse Index)	**Adelson-Velskii and Landis Tree (AVL Tree)**		
	ค่าเฉลี่ย 15 รอบ (มิลลิวินาที)	**ค่าเฉลี่ยต่อคำหลัก (มิลลิวินาที)**	**เวลาเรียงลำดับไฟล์ดรรชนี (มิลลิวินาที)**
การดึงข้อมูลตามดรรชนีแบบกระจาย	30.903	0.059	137.335

ในการแจกแจงแบบปกติ			
การดึงข้อมูลตามดรรชนีแบบกระจายในการแจกแจงแบบเอียงซ้าย	35.2	0.067	
การดึงข้อมูลตามดรรชนีแบบกระจายในการแจกแจงแบบเอียงขวา	35.223	0.067	

การทดลองครั้งแรก มีวัตถุประสงค์เพื่อทดสอบและเปรียบเทียบประสิทธิภาพของเวลาการเข้าถึงตำแหน่งด้วยดรรชนีแบบหนาแน่น (Dense Index) ซึ่งสามารถแบ่งการทดสอบออกเป็นสองลักษณะดังนี้ ลักษณะแรกเป็นการทดทดสอบของแต่ละรอบ โดยประกอบด้วยคำหลัก 524 คำ ซึ่งมีค่าเฉลี่ยของทั้งหมด 15 รอบ หรือเรียกว่าเวลาเฉลี่ย 15 รอบ (Average time of 15 rounds) ผลการทดสอบพบว่า กรณีที่คำหลักมีลักษณะการแจกแจงแบบปกติ เทคนิค AVL Tree มีความเร็วสูงสุด โดยใช้เวลาเฉลี่ยเท่ากับ 2.703 มิลลิวินาที ในขณะที่เทคนิคที่ช้าที่สุดคือการค้นหาเชิงเส้น (LS) ซึ่งใช้เวลาเฉลี่ยเท่ากับ 342,259.155 มิลลิวินาที กรณีที่คำหลักมีลักษณะการแจกแจงแบบเอียงทางซ้าย เทคนิค AVL Tree มีความเร็วสูงสุด โดยใช้เวลาเฉลี่ยเท่ากับ 2.346 มิลลิวินาที ในขณะที่เทคนิคที่ช้าที่สุดคือการค้นหาเชิงเส้น (LS) ซึ่งใช้เวลาเฉลี่ยเท่ากับ 337,220.625 มิลลิวินาที และกรณีที่คำหลักมีลักษณะการแจกแจงแบบเอียงทางขวา เทคนิค AVL Tree มีความเร็วสูงสุด 2.433 มิลลิวินาที ในขณะที่เทคนิคที่ช้าที่สุดคือการค้นหาเชิงเส้น (LS) ซึ่งใช้เวลาเฉลี่ยเท่ากับ 349,457.13 มิลลิวินาที และลักษณะที่สองเป็น การทดสอบการเข้าถึงตำแหน่งด้วยค่าเฉลี่ยต่อคำหลัก (Average per keyword) ผลการวิจัยพบว่า กรณีที่คำหลักมีลักษณะการแจกแจงแบบปกติ เทคนิค AVL Tree มีความเร็วสูงสุด โดยใช้เวลาเฉลี่ยเท่ากับ 0.005 มิลลิวินาที ในขณะที่เทคนิคที่ช้าที่สุดคือการค้นหาเชิงเส้น (LS) ซึ่งใช้เวลาเฉลี่ยเท่ากับ 653.166 มิลลิวินาที กรณีที่คำหลักมีลักษณะการแจกแจงแบบเอียงทางซ้าย เทคนิค AVL Tree มีความเร็วสูงสุด โดยใช้เวลาเฉลี่ยเท่ากับ 0.004 มิลลิวินาที ในขณะที่เทคนิคที่ช้าที่สุดคือการค้นหาเชิงเส้น (LS) ซึ่งใช้เวลาเฉลี่ยเท่ากับ 643.551 มิลลิวินาที และกรณีที่คำหลักมีการแจกแจงแบบเอียงทางขวา เทคนิค AVL Tree มีความเร็วสูงสุด โดยใช้เวลาเฉลี่ยเท่ากับ 0.005 มิลลิวินาที ในขณะที่เทคนิคที่ช้าที่สุดคือการค้นหาเชิงเส้น (LS) ซึ่งใช้เวลาเฉลี่ยเท่ากับ 666.903 มิลลิวินาที

การทดลองครั้งที่สอง มีวัตถุประสงค์เพื่อทดสอบและเปรียบเทียบประสิทธิภาพของเวลาการเข้าถึงตำแหน่งด้วยดรรชนีแบบกระจาย (Sparse Index) ซึ่งสามารถแบ่งการทดสอบออกเป็นสองลักษณะดังนี้ ลักษณะแรกเป็นการทดทดสอบของแต่ละรอบ โดยประกอบด้วยคำหลัก 524 คำ ซึ่งมีค่าเฉลี่ยของทั้งหมด 15 รอบ หรือเรียกว่าเวลาเฉลี่ย 15 รอบ (Average time of 15 rounds) ผลการทดสอบพบว่า กรณีที่คำหลักมีลักษณะการแจกแจงแบบปกติ เทคนิค การค้นหาแบบทวิภาค (Binary search, BS) มีความเร็วสูงสุด โดยใช้เวลาเฉลี่ยเท่ากับ 5.647 มิลลิวินาที ในขณะที่เทคนิคที่ช้าที่สุดคือการค้นหาเชิงเส้น (Linear search, LS) ซึ่งใช้เวลาเฉลี่ยเท่ากับ 528,112.194 มิลลิวินาที กรณีที่คำหลักมีลักษณะการแจกแจงแบบเอียงทางซ้าย เทคนิคการค้นหาแบบทวิภาค (Binary search, BS) มีความเร็วสูงสุด โดยใช้เวลาเฉลี่ยเท่ากับ 3.868 มิลลิวินาที ในขณะที่เทคนิคที่ช้าที่สุดคือการค้นหาเชิงเส้น (Linear search, LS) ซึ่งใช้เวลาเฉลี่ยเท่ากับ 528,112.194 มิลลิวินาที และกรณีที่คำหลักมีลักษณะการแจกแจงแบบเอียงทางขวา เทคนิคการค้นหาแบบทวิภาค (Binary search, BS) มีความเร็วสูงสุด 5.592 มิลลิวินาที ในขณะที่เทคนิคที่ช้าที่สุดคือการค้นหาเชิงเส้น (Linear search, LS) ซึ่งใช้เวลาเฉลี่ยเท่ากับ 528,179.767 มิลลิวินาที และลักษณะที่สองเป็นการทดสอบการเข้าถึงตำแหน่งด้วยค่าเฉลี่ยต่อคำหลัก (Average per keyword) ผลการวิจัยพบว่า กรณีที่คำหลักมีลักษณะการแจกแจงแบบปกติ เทคนิคการค้นหาแบบทวิภาค (Binary search, BS) มีความเร็วที่สูงสุด โดยใช้เวลาเฉลี่ยเท่ากับ 0.11 มิลลิวินาที ในขณะที่เทคนิคที่ช้าที่สุดคือต้นไม้ค้นหาแบบทวิภาค (Binary search Tree, BST) ซึ่งใช้เวลาเฉลี่ยเท่ากับ 0.307 มิลลิวินาที กรณีที่คำหลักมีลักษณะการแจกแจงแบบเอียงทางซ้าย เทคนิคการค้นหาแบบทวิภาค (Binary search, BS) มีความเร็วสูงสุด โดยใช้เวลาเฉลี่ยเท่ากับ 0.007 มิลลิวินาที ในขณะที่เทคนิคที่ช้าที่สุดคือต้นไม้ค้นหาแบบทวิภาค (Binary search Tree, BST) ซึ่งใช้เวลาเฉลี่ยเท่ากับ 0.317 มิลลิวินาที และกรณีที่คำหลักมีการแจกแจงแบบเอียงทางขวา เทคนิคการค้นหาแบบทวิภาค (Binary search, BS) มีความเร็วสูงสุด โดยใช้เวลาเฉลี่ยเท่ากับ 0.11 มิลลิวินาที ในขณะที่เทคนิคที่ช้าที่สุดคือต้นไม้ค้นหาแบบทวิภาค (Binary search Tree, BST) ซึ่งใช้เวลาเฉลี่ยเท่ากับ 0.302 มิลลิวินาที

บทที่ 5

สรุปผลการวิจัย อภิปรายผล และข้อเสนอแนะ

การวิจัยนี้นำเสนอการเพิ่มประสิทธิภาพในการสำรวจและเข้าถึงข้อมูล โดยใช้เทคนิคการสร้างดรรชนี สำหรับชุดข้อมูลขนาดใหญ่ ภายใต้รูปแบบวัตถุของจาวาสคริปต์ โดยการนำประยุกต์เทคนิคการจัดทำดรรชนีแบบหนาแน่น (Dense Index) และดรรชนีแบบกระจาย (Sparse Index) เพื่อเพิ่มประสิทธิภาพการสำรวจและเข้าถึงข้อมูล การค้นพบนี้แสดงให้เห็นว่าวิธีการสร้างดรรชนีใหม่ช่วยลดเวลาในการสำรวจและเข้าถึงข้อมูลลดลงได้อย่างมาก นอกจากนี้ การทดลองในทั้งสามช่วงขนาดส่วน (Segment size) ตั้งแต่ช่วง 100 ถึง 1,000, 1,000 ถึง 10,000 และ 10,000 ถึง 200,000 ผลลัพธ์พิสูจน์ให้เห็นว่าการเปลี่ยนแปลงขนาดส่วนไม่ได้ส่งผลต่อประสิทธิภาพทางด้านเวลาในการเข้าถึงคำหลัก (Keyword) ซึ่งเป็นผลมาจากการเข้าถึงหน่วยความจำเข้าถึงแบบสุ่ม (Random Access Memory, RAM) ที่ใกล้เคียงกันเมื่อเทียบกับขนาดส่วน (Segment size) ทั้งนี้ขึ้นอยู่กับประสิทธิภาพของระบบคอมพิวเตอร์ ดังนั้น ไม่ว่าขนาดส่วน (Segment size) จะอยู่ในช่วงใด ดรรชนีแบบหนาแน่น (Dense Index) และดรรชนีแบบกระจาย (Sparse Index) ยังคงสามารถจัดการกับหน่วยความจำและการเข้าถึงคำหลัก (Keyword) ได้อย่างรวดเร็ว นอกจากนี้ ดรรชนีแบบหนาแน่น (Dense Index) สามารถเพิ่มประสิทธิภาพทางด้านเวลาในการเข้าถึงกับลักษณะข้อมูลแบบหนึ่งต่อหนึ่ง (One-to-One) เฉลี่ยต่อคำหลักเท่ากับ 382.196 มิลลิวินาที เมื่อเทียบกับการเข้าถึงข้อมูลของกรณีไม่มีดรรชนี (Non Index) เฉลี่ยต่อคำหลักเท่ากับ 26,869.218 มิลลิวินาที ในขณะที่ดรรชนีแบบกระจาย (Sparse Index) สามารถเพิ่มประสิทธิภาพทางด้านเวลาในการเข้าถึงกับลักษณะข้อมูลแบบหนึ่งต่อกลุ่ม (One-to-Many) เฉลี่ยต่อคำหลักเท่ากับ 854.662 มิลลิวินาที เมื่อเทียบกับการเข้าถึงข้อมูลของกรณีไม่มีดรรชนี (Non Index) เฉลี่ยต่อคำหลักเท่ากับ 55,197.734 มิลลิวินาที การค้นพบนี้ชี้ให้เห็นว่าแนวทางการจัดทำดรรชนีแบบหนาแน่น (Dense Index) สามารถลดเวลาในการเข้าถึงข้อมูลถึง 98.57% จากวิธีการไม่มีดรรชนีและดรรชนีแบบกระจาย (Sparse Index) สามารถลดเวลาในการเข้าถึงข้อมูลถึง 98.45% จากวิธีการไม่มีดรรชนี ทั้งสองวิธีการดรรชนีของผู้วิจัยสามารถช่วยเพิ่มการสำรวจและเข้าถึงในชุดฐานข้อมูลที่ไม่ใช่เชิงสัมพันธ์ (Not only Structured Query Language, NoSQL) ที่จัดเก็บอยู่ภายใต้รูปแบบวัตถุของจาวาสคริปต์ โดยเฉพาะอย่างยิ่งเมื่อจำลองชุดข้อมูลขนาดใหญ่ การศึกษาในอนาคตสามารถขยายการวิจัยนี้ได้โดยการทดสอบประสิทธิภาพของเทคนิคการจัดทำดรรชนี กับข้อมูลประเภทต่าง ๆ และชุด

ข้อมูลขนาดต่าง ๆ เพื่อกำหนดความสามารถในการปรับขนาดและความสามารถทั่วไปในการทำงานในอนาคต การปรับปรุงอัลกอริทึมการสำรวจและเข้าถึงตำแหน่งข้อมูลในไฟล์ดรรชนีแบบหนาแน่น (Dense Index) อาจเป็นส่วนที่มีศักยภาพในการตรวจสอบ เนื่องจากไฟล์ดรรชนีแบบหนาแน่น มักจะมีข้อมูลจำนวนมาก กระบวนการค้นหาจึงใช้เวลานานและไม่มีประสิทธิภาพ การลดเวลาที่จำเป็นสำหรับกระบวนการนี้ สามารถปรับปรุงประสิทธิภาพโดยรวมของแนวทางการจัดทำดรรชนีแบบหนาแน่น (Dense Index) ได้อย่างมาก ดังนั้น วิธีการเพิ่มประสิทธิภาพอัลกอริทึมของการสำรวจและเข้าถึงสำหรับการจัดทำดรรชนีแบบหนาแน่น (Dense Index) อาจเป็นช่องทางที่น่าสนใจสำหรับงานในอนาคต นอกจากนี้ เราดำเนินการวิจัยเพิ่มเติมเกี่ยวข้องกับการเปรียบเทียบประสิทธิภาพทางเวลาในการเข้าถึงตำแหน่งดรรชนีแบบหนาแน่น (Dense Index) และดรรชนีแบบกระจาย (Sparse Index) โดยการนำประยุกต์เทคนิคการค้นหาแบบเรียงลำดับ (Linear Search, LS) การค้นหาแบบทวิภาค (Binary Search, BS) ต้นไม้ค้นหาแบบทวิภาค (Binary Search Tree, BST) และ Adelson–Velskii Landis (AVL Tree) ทดสอบร่วมกับลักษณะคำหลักที่มีการแจกแจงทั้งหมดสามรูปแบบ ได้แก่ การแจกแจงแบบปกติ (Normal distribution) การแจกแจงแบบเอียงซ้าย (Left skewed distribution) และการแจกแจงแบบเอียงขวา (Right skewed distribution) ผลการทดลองแสดงให้เห็นว่า ดชชนีแบบหน้าแน่น ((Dense Index) สำหรับคำหลักที่มีการแจงแบบปกติพบว่าเทคนิค AVL Tree เป็นเทคนิคที่เข้าถึงตำแหน่งได้เร็วที่สุด โดยใช้เวลาเฉลี่ยเท่ากับ 0.005 มิลลิวินาทีต่อคำหลัก เทคนิคที่ช้าที่สุดคือเทคนิค LS โดยใช้เวลาเฉลี่ยเท่ากับ 653.166 มิลลิวินาทีต่อคำหลัก สำหรับคำหลักที่มีการกระจายเอียงซ้าย เทคนิค AVL Tree ยังคงเป็นวิธีที่เร็วที่สุด โดยใช้เวลาเฉลี่ยเท่ากับ 0.004 มิลลิวินาทีต่อคำหลัก และเทคนิค LS ช้าที่สุด มีค่าเฉลี่ย 643.551 มิลลิวินาทีต่อคำหลัก ในขณะที่คำหลักมีการกระจายแบบเอียงขวา เทคนิค AVL Tree ยังคงเร็วที่สุด โดยใช้เวลาเฉลี่ยเท่ากับ 0.005 มิลลิวินาทีต่อคำหลัก และเทคนิค LS ช้าที่สุด โดยใช้เวลาเฉลี่ยเท่ากับ 666.903 มิลลิวินาทีต่อคำหลัก ในขณะที่ดรรชนีแบบกระจาย (Sparse Index) สำหรับคำหลักที่มีแจกแจงแบบปกติพบว่าเทคนิค BS เป็นเทคนิคที่มีความเร็วมากที่สุด โดยใช้เวลาเฉลี่ยเท่ากับ 0.011 มิลลิวินาทีต่อคำหลัก และเทคนิคที่ช้าที่สุดคือเทคนิค LS โดยใช้เวลาเฉลี่ยเท่ากับ 1007.848 มิลลิวินาทีต่อคำหลัก สำหรับคำหลักที่มีการกระจายเอียงซ้าย เทคนิค BS ยังคงเป็นเทคนิคที่เร็วที่สุด โดยใช้เวลาเฉลี่ยเท่ากับ 0.007 มิลลิวินาทีต่อคำหลัก เทคนิคที่ช้าที่สุดคือเทคนิค LS โดยใช้เวลาเฉลี่ยเท่ากับ 1008.117 มิลลิวินาทีต่อคำหลัก ในกรณีของการกระจายแบบเอียงขวา เทคนิค BS เป็นเทคนิคที่เร็วที่สุด โดยใช้เวลาเฉลี่ยเท่ากับ 0.011 มิลลิวินาทีต่อคำหลัก และเทคนิค LS เป็นเทคนิคที่ช้าที่สุด โดยใช้เวลาเฉลี่ยเท่ากับ 1007.977 มิลลิวินาทีต่อคำหลัก

ดังนั้น ผลการทดลองแสดงให้เห็นว่าเทคนิค AVL Tree มีประสิทธิภาพสูงในการจัดการดรรชนีแบบหนาแน่น (Dense Index) เหมาะสมกับการจัดการคำหลักจำนวนมาก (เช่น 1,000,000 รายการ) ด้วยเวลาเฉลี่ยในการเข้าถึงที่ 0.005 มิลลิวินาทีต่อคำหลัก ความสามารถนี้เกิดจากคุณสมบัติการสมดุลตัวเองของ AVL Tree ที่ช่วยให้การค้นหาในชุดข้อมูลขนาดใหญ่เป็นไปอย่างรวดเร็วและมีประสิทธิภาพ ในทางตรงกันข้ามเทคนิค Binary Search แสดงประสิทธิภาพที่ดีเมื่อใช้กับดรรชนีแบบกระจาย (Sparse Index) เหมาะสำหรับจำนวนคำหลักที่จำกัด (เช่น 5,000 รายการ) โดยมีเวลาเฉลี่ยในการเข้าถึง 0.011 มิลลิวินาทีต่อคำหลัก ความสามารถนี้มาจากการที่ Binary Search สามารถทำการค้นหาต่อเนื่องในข้อมูลที่มีการจัดเรียงไว้ล่วงหน้า การศึกษานี้ชี้ให้เห็นถึงความสำคัญของการเลือกโครงสร้างข้อมูลที่เหมาะสมกับลักษณะของดรรชนีและข้อมูล เพื่อเพิ่มประสิทธิภาพในการเข้าถึงและจัดการข้อมูล โดยเฉพาะ AVL Tree ที่เหมาะสมอย่างยิ่งสำหรับข้อมูลหนาแน่นและจำนวนมาก ขณะที่ Binary Search มีประสิทธิภาพในการจัดการกับข้อมูลจำนวนจำกัดและกระจาย ส่วนเทคนิค Linear Search (LS) มีประสิทธิภาพเวลาช้าที่สุด ในการเข้าถึงตำแหน่งต่อคำหลักของดรรชนีทั้งสอง ซึ่งเป็นผลมาจากวิธีการค้นหาแบบตรงไปตรงมาโดยไม่มีการใช้การจัดเรียงข้อมูล อีกทั้งการค้นหาด้วยเทคนิค LS จำเป็นต้องทำการตรวจสอบแต่ละคำหลักทีละครั้งจนกว่าจะพบข้อมูลที่ต้องการ ซึ่งทำให้เวลาที่ใช้ในการค้นหาเพิ่มขึ้นตามจำนวนข้อมูล จึงไม่เหมาะกับชุดข้อมูลขนาดใหญ่ สำหรับงานวิจัยในอนาคตสามารถขยายขอบเขตจากผลการศึกษานี้ โดยการสร้างดรรชนีรอง (Secondary index) สำหรับประเภทข้อมูลและชุดข้อมูลที่ซับซ้อนมากขึ้น เพื่อกำหนดขีดจำกัดด้านความสามารถในการขยายขนาดและการประยุกต์ใช้

บรรณานุกรม

Abdulkadhem, A. A. และAl-Assadi, T. A. (2019). An important landmarks construction for a GIS-Map based on indexing of dolly images. **Indonesian Journal of Electrical Engineering and Computer Science**, 15(1), 451-459.

Abdulsada, A. I., Honi, D. G. และAl-Darraji, S. (2021). Efficient multi-keyword similarity search over encrypted cloud documents. **Indones. J. Electr. Eng. Comput. Sci**, 23(1), 510-518.

Alamelu, R. และPrabu, K. (2022). Hybridization of Pigeon inspired with glowworm'swarm optimization based clustering technique in wireless sensor networks. **Microprocessors and Microsystems**, 91, 104528.

Alqatawneh, A. (2022). Orthogonal frequency division multiplexing system with an indexed-pilot channel estimation. **Indonesian Journal of Electrical Engineering and Computer Science**, 26(2), 808. จาก https://doi.org/10.11591/ijeecs.v26.i2.pp808-818

Althaf Ali, A. และMahammad Shafi, R. (2019). Test-framework: performance profiling and testing web search engine on non factoid queries. **Indonesian Journal of Electrical Engineering and Computer Science**, 14(3).

Asri, Y., Kuswardani, D., Yosrita, E. และWullur, F. H. (2021). Clusterization of customer energy usage to detect power shrinkage in an effort to increase the efficiency of electric energy consumption. **Indonesian Journal of Electrical Engineering and Computer Science**, 22(1), 10-17.

Baykasoğlu, A. และOzsoydan, F. B. (2018). Dynamic optimization in binary search spaces via weighted superposition attraction algorithm. **Expert Systems with Applications**, 96, 157-174.

Belazzougui, D., Cunial, F., Kärkkäinen, J. และMäkinen, V. (2020). Linear-time string indexing and analysis in small space. **ACM Transactions on Algorithms (TALG)**, 16(2), 1-54.

Bertalya, B., Prihandoko, P., Setyowati, L., Irawan, F. I. และIrlianti, S. R. (2021). Formulation

of city health development index using data mining. **Indonesian Journal of Electrical Engineering and Computer Science**, 23(1), 362. doi:https://doi.org/10.11591/ijeecs.v23.i1.pp362-369.

Buldas, A. Draheim, D. Gault, M. Laanoja, R. Nagumo, T. Saarepera, M. และคณะ. (2022). An ultra-scalable blockchain platform for universal asset tokenization: design and implementation. **IEEE Access**, 10, 77284-77322.

Chang, J. Xiao, L. Huo, Z. Zhou, B. Ruan, L. Wang, H. และคณะ. (2017). Optimization of Index-Based Method of Metadata Search for Large-Scale File Systems. In **2017 10th International Symposium on Computational Intelligence and Design (ISCID)** (pp 476-481).

Chen, D., Wu, C. Q., Shen, W. และZhang, Y. (2021). On a Small File Merger for Fast Access and Modifiability of Small Files in HDFS. In **2021 IEEE/ACS 18th International Conference on Computer Systems and Applications (AICCSA)** (pp 1-8).

Chopade, R. และPachghare, V. (2020). MongoDB indexing for performance improvement. In **ICT Systems and Sustainability** (pp 529-539).

Cormen, T. H., Leiserson, C. E., Rivest, R. L. และStein, C. (2009). **Introduction to Algorithms** (พิมพ์ครั้งที่ 3). Cambridge, Mass.: Mit Press.

Dittrich, J., Nix, J. และSchön, C. (2021). The next 50 years in database indexing or: the case for automatically generated index structures. **Proceedings of the VLDB Endowment**, 15(3), 527-540.

Elhoussaine, Z. (2020). A fuzzy neighborhood rough set method for anomaly detection in large scale data. **IAES International Journal of Artificial Intelligence**, 9(1), 1.

Elmasri, R. และNavathe, S. B. (2011). **Fundamentals of Database Systems** (พิมพ์ครั้งที่ 6): Pearson Education, Inc.

Fathy, Y., Barnaghi, P. และTafazolli, R. (2018). Large-scale indexing, discovery, and ranking for the Internet of Things (IoT). **ACM Computing Surveys (CSUR)**, 51(2), 1-53.

Ferragina, P. และVinciguerra, G. (2019). The PGM-index: a multicriteria, compressed and learned approach to data indexing. **arXiv preprint arXiv:1910.06169**.

Fu, J., Wang, N., Cui, B. และBhargava, B. K. (2021). A practical framework for secure

document retrieval in encrypted cloud file systems. **IEEE Transactions on Parallel and Distributed Systems**, 33(5), 1246–1261.

Gayathiri, N., Jaspher, D. D. และNatarajan, A. (2019). Big Data retrieval techniques based on hash indexing and Mapreduce approach with NoSQL Database. In **2019 international conference on advances in computing and communication engineering (ICACCE)** (pp 1–8).

Gebeyehu, S., Wolde, W. และS. Shibeshi, Z. (2023). Information extraction model from Ge'ez texts. **Indonesian Journal of Electrical Engineering and Computer Science**, 30(2), 787–795. doi:https://doi.org/10.11591/ijeecs.v30.i2.pp787–795.

GeeksforGeeks. (n.d.). **Indexing in Databases**. สืบค้นเมื่อ ตุลาคม 2565, จาก https://www.geeksforgeeks.org/indexing–in–databases–set–1/

Gheraibia, Y., Moussaoui, A., Djenouri, Y., Kabir, S. และYin, P. Y. (2016). Penguins search optimisation algorithm for association rules mining. **Journal of computing and information technology**, 24(2), 165–179.

Goodrich, M. T. และTamassia, R. (2014). **Algorithm Design and Applications** (พิมพ์ครั้งที่ 1): Wiley.

Gopinathan, D. และAsawa, K. (2017). New Path Based Index Structure for Processing CAS Queries over XML Database. **Journal of computing and information technology**, 25(3), 211–225.

Guellouma, Y. และCherroun, H. (2016). Efficient implementation for deterministic finite tree automata minimization. **Journal of computing and information technology**, 24(4), 311–322.

Han, J., Haihong, E., Le, G. และDu, J. (2011). Survey on NoSQL database. In **2011 6th international conference on pervasive computing and applications** (pp 363–366).

Hashem, I. A. T. และPillai, T. R. (2020). A novel feature engineering algorithm for air quality datasets. **Indonesian Journal of Electrical Engineering and Computer Science**, 19(3), 1444–1451.

IBM. (n.d.). **NoSQL Databases**. สืบค้นเมื่อ ตุลาคม 2565, จาก

https://www.ibm.com/topics/nosql-databases

Ibrahim, H., Sani, N. F. M. และYaakob, R. (2020). Analyses of indexing techniques on uncertain data with high dimensionality. **IEEE Access**, 8, 74101-74117.

Jiang, L. และZhao, Z. (2022). JSONSki: streaming semi-structured data with bit-parallel fast-forwarding. In (pp 200-211).

Jin, P., Zhuang, X., Luo, Y. และLu, M. (2021). Exploring index structures for zoned namespaces SSDs. In **2021 IEEE International Conference on Big Data (Big Data)** (pp 5919-5922).

Kareem, S. W., Yousif, R. Z. และJihad Abdalwahid, S. M. (2020). An approach for enhancing data confidentiality in Hadoop. **Indonesian Journal of Electrical Engineering and Computer Science**, 20(3), 1547-1555. doi:https://doi.org/10.11591/ijeecs.v20.i3.pp1547-1555.

Knuth, D. E. (1998). **The art of computer programming, volume 3: (2nd ed.) sorting and searching** (พิมพ์ครั้งที่ 2): Addison Wesley Longman Publishing Co., Inc.

Komorowski, M. และTrzciński, T. (2019). Random binary search trees for approximate nearest neighbour search in binary spaces. **Applied Soft Computing**, 79, 87-93.

Kraska, T., Beutel, A., Chi, E. H., Dean, J. และPolyzotis, N. (2018). The case for learned index structures. In **Proceedings of the 2018 international conference on management of data** (pp 489-504).

Kuan, Y.-H., Chang, Y.-H., Chen, T.-Y., Huang, P.-C. และLam, K.-Y. (2016). Space-efficient index scheme for PCM-based multiversion databases in cyber-physical systems. **ACM Transactions on Embedded Computing Systems (TECS)**, 16(1), 1-26.

Kühn, R., Biebert, D., Hakert, C., Chen, J.-J. และTeubner, J. (2023). Towards Data-Based Cache Optimization of B+-Trees. In **Proceedings of the 19th International Workshop on Data Management on New Hardware** (pp 63-69).

Lee, D., Althoff, A., Richmond, D. และKastner, R. (2017). A streaming clustering approach using a heterogeneous system for big data analysis. In **2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)** (pp 699-706).

Ma, R., Zheng, D. และYan, L. (2020). Fast Online Similarity Search for Uncertain Time Series. **Journal of computing and information technology**, 28(1), 1–17.

Ma, R., Zhu, X. และYan, L. (2020). A Hybrid Approach for Clustering Uncertain Time Series. **Journal of computing and information technology**, 28(4), 255–267.

Ma, Y., Liu, D., Scott, G., Uhlmann, J. และShyu, C.–R. (2017). In–memory distributed indexing for large–scale media data retrieval. In **2017 IEEE International Symposium on Multimedia (ISM)** (pp 232–239).

Meier, A., Kaufmann, M., Meier, A. และKaufmann, M. (2019). Nosql databases. **SQL & NoSQL Databases: Models, Languages, Consistency Options and Architectures for Big Data Management**, 201–218.

MongoDB. (n.d.). **Indexes**. สืบค้นเมื่อ ตุลาคม 2565, จาก https://www.mongodb.com/docs/manual/indexes/

Oujdi, S., Belbachir, H. และBoufares, F. (2019). C4. 5 decision tree algorithm for spatial data, alternatives and performances. **Journal of computing and information technology**, 27(3), 29–43.

Panatula, G., Kumar, K. S., Geetha, D. E. และKumar, T. V. S. (2019). Performance evaluation of cloud service with hadoop for twitter data. **Indonesian Journal of Electrical Engineering and Computer Science**, 13(1), 392. จาก https://doi.org/10.11591/ijeecs.v13.i1.pp392–404

Prasad, S. และKulkarni, S. (2021). Quality and energy optimized scheduling technique for executing scientific workload in cloud computing environment. **Indonesian Journal of Electrical Engineering and Computer Science**, 21(2).

Priyadarshini, R. R. และSivakumar, N. (2021). Cluster head selection based on minimum connected dominating set and bi–partite inspired methodology for energy conservation in WSNs. **Journal of King Saud University–Computer and Information Sciences**, 33(9), 1132–1144.

Qader, M. A., Cheng, S. และHristidis, V. (2018). A comparative study of secondary indexing techniques in LSM–based NoSQL databases. In **Proceedings of the 2018 International Conference on Management of Data** (pp 551–566).

Qiu, R., Ming, Y., Hong, Y., Li, H. และYang, T. (2023). Wind-Bell Index: Towards Ultra-Fast Edge Query for Graph Databases. In **2023 IEEE 39th International Conference on Data Engineering (ICDE)** (pp 2090-2098).

S, M. และMb, S. P. (2020). Indexing intelligence using benchmark classifier. **Indonesian Journal of Electrical Engineering and Computer Science**, 18(1), 179-187. doi:https://doi.org/10.11591/ijeecs.v18.i1.pp179-187.

Sedgewick, R. และWayne, K. X. (2011). **Algorithms** (พิมพ์ครั้งที่ 4): Addison-wesley professional.

Shen, H., Xue, L., Wang, H., Zhang, L. และZhang, J. (2021). B+-Tree Based Multi-Keyword Ranked Similarity Search Scheme Over Encrypted Cloud Data. **IEEE Access**, 9, 150865-150877.

Shin, Y., Ahn, J. และIm, D.-H. (2022). Join optimization for inverted index technique on relational database management systems. **Expert Systems with Applications**, 198, 116956.

Silberschatz, A., Galvin, P. B.,Gagne, G. (2013). **Operating System Concepts** (พิมพ์ครั้งที่ 9): John Wiley & Sons.

Silwattananusarn, T. และKulkanjanapiban, P. (2022). A text mining and topic modeling based bibliometric exploration of information science research. **IAES International Journal of Artificial Intelligence**, 11(3), 1057.

Song, J., Shen, H. T., Wang, J., Huang, Z., Sebe, N. และWang, J. (2016). A distance-computation-free search scheme for binary code databases. **IEEE Transactions on Multimedia**, 18(3), 484-495.

Stanly Wilson, S. R. (2019). Twitter data analysis using hadoop ecosystems and apache zeppelin. **Indonesian Journal of Electrical Engineering and Computer Science**, 16(3), 1490-1498.

Tan, L. K. และLim, K. C. (2019). Fast surveillance video indexing & retrieval with WiFi MAC address tagging. **Indonesian Journal of Electrical Engineering and Computer Science**, 16(1), 473-481. doi:https://doi.org/10.11591/ijeecs.v16.i1.pp473-481.

Wang, Q., Zhang, X., Qin, J., Ma, J. และHuang, X. (2023). A verifiable symmetric searchable

encryption scheme based on the AVL tree. **The Computer Journal**, 66(1).

Wang, Z., Wang, Q., Wang, P., Palpanas, T. และWang, W. (2023). Dumpy: A compact and adaptive index for large data series collections. **Proceedings of the ACM on Management of Data**, 1(1), 1–27.

Wu, H. Zhu, Y. Wang, C. Hou, J. Li, M. Xue, Q. และคณะ. (2017). A performance–improved and storage–efficient secondary index for big data processing. In **2017 IEEE International Conference on Smart Cloud (SmartCloud)** (pp 161–167).

Wu, J., Xiong, J., Dai, H., Wang, Y. และXu, C. (2022). MIX–RS: A multi–indexing system based on HDFS for remote sensing data storage. **Tsinghua Science and Technology**, 27(6).

Yuan, J. และLiu, X. (2012). A novel index structure for large scale image descriptor search. In **2012 19th IEEE International Conference on Image Processing** (pp 1937–1940).

Yusof, M. K. และMan, M. (2017). Efficiency of JSON for data retrieval in big data. **Indonesian Journal of Electrical Engineering and Computer Science**, 7(1), 250–262.

Zeffora, J. และShobarani, S. (2022). Optimizing random forest classifier with Jenesis–index on an imbalanced dataset. **Indonesian Journal of Electrical Engineering and Computer Science**, 26(1), 505–511. doi:https://doi.org/10.11591/ijeecs.v26.i1.pp505–511.

Zhao, Y.–S. และZeng, Q.–A. (2018). Secure and efficient product information retrieval in cloud computing. **IEEE Access**, 6, 14747–14754.

Zineddine, K., Amine, F. M. และAdeel, A. (2021). Indexing Multimedia Data with an Extension of Binary Tree––Image Search by Content. **International Journal of Informatics and Applied Mathematics**, 1(1), 47–55.

มหาวิทยาลัยพะเยา
UNIVERSITY OF PHAYAO

ประวัติผู้วิจัย

ชื่อ-สกุล	จิรวัฒน์ ดวงแก้ว
วัน เดือน ปี เกิด	01 กรกฎาคม 2540
สถานที่เกิด	โรงพยาบาลบ้านฉาง
วุฒิการศึกษา	พ.ศ 2563 วท.บ. (วิทยาการคอมพิวเตอร์), มหาวิทยาลัยราชภัฏรำไพพรรณี, จันทบุรี
ที่อยู่ปัจจุบัน	24/2 ซอยคีรี ตำบลห้วยโป่ง อำเภอเมืองระยอง จังหวัดระยอง 21150
ผลงานตีพิมพ์	Duangkaew, J., Srisungsittisunt, B., Witayarat, A., Boonthep, N., Phumsaranakhom, P., & Chaiwongsai, J. (2023). Enhancing Data Retrieval Efficiency in Geospatial Surveys Using Indexing Techniques for Large-Scale JavaScript Object Notation Datasets. The Journal of Spatial Innovation Development, 4(3), 27-45.