การพัฒนาอัลกอลิทึมและการจำลองการจัดการแคชที่ใช้เก็บ URL

THE DEVELOPMENT OF AN ALGORITHM FOR URL CACHE

กฤษฎา กัปปิยบุตร
ทีฆายุ แก้วฉิม
ณัฐพล ดำรงรัตน์นุวงศ์

ปัญหาพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต
ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์
คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2543

THE DEVELOPMENT OF AN ALGORITHM FOR URL CACHE

KRITSADA KUPPIYABUTRA

TEEKAYU KAEWCHIM

NUTHAPHON DAMRONGRATNUWONG

A SPECIAL PROJECT SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF BACHELOR OF SCIENCE

DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCES

FACULTY OF SCIENCE

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LARDKRABANG

ACADEMIC YEAR 2000

ปัญหาพิเศษเรื่อง	การพัฒนาอัลกอลิทึมและการจำลองการจัดการแคชที่ใช้เก็บURL	
	THE DEVELOPMENT OF AN ALGORITHM FOR URL CACHE	
ชื่อนักศึกษา	นายกฤษฎา กัปปิยบุตร	40056003
	นายฑีฆายุ แก้วฉิม	40056022
	นายณัฐพล ดำรงรัตน์นุวงศ์	40056027
ภาควิชา	คณิตศาสตร์และวิทยาการคอมพิวเตอร์	
ปีการศึกษา	2543	
อาจารย์ที่ปรึกษา	อาจารย์ชริญญา ศิริมังคลานุรักษ์	

ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง อนุมัติให้นับปัญหาพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร วิทยาศาสตรบัณฑิต สาขาวิทยาการคอมพิวเตอร์ ประจำปีการศึกษา 2543

คณะกรรมการสอบ		ลายมือชื่อ
ประธานกรรมการ	อาจารย์วีระชัย ตันยะสิทธิ์	
กรรมการ	ผู้ช่วยศาสตราจารย์ธีรวัฒน์ ประกอบผล	
กรรมการและอาจารย์ที่ปรึกษา	อาจารย์ชริญญา ศิริมังคลานุรักษ์	

(ผู้ช่วยศาสตราจารย์ไพรบูลย์ พันธรักพงษ์)
หัวหน้าภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์

สารบัญ

หน้า

สารบัญภาพ

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในปัจจุบันนี้ข้อมูลต่างมีความสำคัญเป็นอย่างมากในการดำเนินกิจการ ให้ได้ผลสำเร็จดังที่คาดไว้ ดังนั้นการที่จะได้มาซึ่งข้อมูลที่รวดเร็วจึงเป็นสิ่งที่มีความสำคัญเช่นกัน ในปัจจุบันการหาข้อมูลที่รวดเร็วและสะดวกที่สุด มิใช่ในห้องสมุดหรือตามศูนย์หนังสือ แต่สามารถค้นหาได้จาก Internet ซึ่งในปัจจุบันเป็นแหล่งรวบรวมข้อมูลที่หลากหลาย ตามปกติเมื่อเราเรียกใช้ข้อมูลผ่าน Internet ข้อมูลจะถูกถ่ายโอนจาก Web Server มายังเครื่องที่เรียกใช้ข้อมูล ข้อมูลจาก Web Server ที่เคยค้นหามาแล้วจะถูกเก็บอยู่ใน Cache ของเครื่องที่เรียกใช้ข้อมูล เพื่อให้การเข้าถึงข้อมูลครั้งต่อไปรวดเร็วยิ่งขึ้น เนื่องจากสามารถเรียกข้อมูลที่เคยเก็บไว้มาแสดงผลบนหน้าจอ โดยไม่ต้องไปนำข้อมูลมาจาก Web Server โดยตรง ดังนั้นถ้าเราพัฒนาวิธีการเก็บข้อมูลที่เคยเรียกใช้มาแล้วให้มีประสิทธิภาพมากขึ้นก็จะทำให้การเรียกใช้ข้อมูลรวดเร็วยิ่งขึ้น

1.2 วัตถุประสงค์ของการทำปัญหาพิเศษ

เนื่องจากการไปเรียกข้อมูลจาก Web Server โดยตรงทุกครั้งจะเป็นการเสียเวลา ดังที่ได้กล่าวไว้ในหัวข้อ 1.1 การเรียกข้อมูลของ Web ที่เคยไปมาแล้ว สามารถทำได้รวดเร็วขึ้น โดยอาศัยการพัฒนาโครงสร้างข้อมูล (Data Structure) ที่ใช้ในการจัดเก็บURL เมื่อ URLถูกจัดเก็บอย่างเป็นระบบใน Cache โดยใช้โครงสร้างข้อมูลที่ได้คิดค้นขึ้น จะทำให้การค้นหาไฟล์ HTML ของ URL ที่เคยถูกเรียกใช้มาแล้วรวดเร็วกว่าการไปเอาข้อมูลโดยตรงจาก Web Server

1.3 สมมติฐานของการศึกษา

การใช้โครงสร้างข้อมูลแบบต้นไม้ในการจัดเก็บ URL อย่างเป็นระบบจะทำให้การค้นหาไฟล์ HTML ของ URL ที่เคยถูกเรียกใช้ใน Cache เพื่อนำมาแสดงผล ทำได้รวดเร็วยิ่งขึ้น

1.4 ขอบเขตของการศึกษา

1.4.1 ศึกษาและเข้าใจในทฤษฎีของระบบโครงสร้างข้อมูลในการเก็บ URL และไฟล์ HTML

1.4.2 วิเคราะห์ระบบโครงสร้างข้อมูลแบบต่าง ๆ ที่เหมาะสมกับทำงาน

1.4.3 วิเคราะห์หาความซับซ้อนของอัลกอริทึม (Complexity of Algorithms) ของระบบโครงสร้างข้อมูลที่เลือกใช้

1.4.4 เขียนโปรแกรมจำลองการทำงานด้วยภาษา Delphi 5.0

1.5 ขั้นตอนในการดำเนินงาน

1.5.1 ศึกษาโครงสร้างข้อมูลแบบต่างๆ

1.5.2 ศึกษาการวิเคราะห์หาความซับซ้อนของอัลกอริทึม

1.5.3 ศึกษาเรื่องต่าง ๆ ที่เกี่ยวกับ Internet

1.5.4 ศึกษาภาษาที่เหมาะสมที่จะใช้เขียนโปรแกรมจำลองการทำงาน

1.5.5 ออกแบบโครงสร้างข้อมูลที่ใช้ในการจัดเก็บและค้นหา URL

1.5.6 วิเคราะห์หาความซับซ้อนของอัลกอริทึมของโครงสร้างข้อมูลที่ออกแบบมา

1.5.7 เขียนโปรแกรมจำลองการทำงานของโครงสร้างข้อมูลที่ได้ออกแบบไว้

1.5.8 จัดทำโปรแกรมในส่วนติดต่อกับผู้ใช้

1.5.9 ทดสอบโปรแกรม โดยใช้ข้อมูลตัวอย่าง

1.5.10 ทำการวิเคราะห์และสรุปผลการทดสอบ

1.5.11 จัดทำเอกสารและเตรียมเสนอผลงาน

1.6 ข้อตกลงเบื้องต้น

ขนาดของ Cache ที่จะจัดเก็บมีขนาดจำกัดอยู่เพียง 1 MB

1.7 ข้อจำกัดของการศึกษา

ขนาดของ Cache ที่จะจัดเก็บมีขนาดจำกัดอยู่เพียง 1 MB

ตารางที่ 1.1 แสดงระยะเวลาในการทำงานแต่ละช่วง

	มิ.ย	ก.ค	ส.ค	ก.ย	ต.ค	พ.ย	ธ.ค
ศึกษาโครงสร้างข้อมูลแบบต่างๆ							
ศึกษาการวิเคราะห์ความซับซ้อนของอัลกอริทึม							
ศึกษาเรื่องต่าง ๆ ที่เกี่ยวกับ Internet							
ศึกษาภาษาที่เหมาะสมที่จะใช้เขียนโปรแกรมจำลองการทำงาน							
ออกแบบโครงสร้างข้อมูลที่ใช้ในการจัดเก็บและค้นหา URL							
วิเคราะห์หาความซับซ้อนของอัลกอริทึมของโครงสร้างข้อมูล							
เขียนโปรแกรมจำลองการทำงานของโครงสร้างข้อมูลที่ได้ออกแบบไว้							
จัดทำโปรแกรมในส่วนติดต่อผู้กับใช้							
ทดสอบโปรแกรม โดยใช้ข้อมูลชุดตัวอย่าง							
ทำการวิเคราะห์และสรุปผลการทดสอบ							
จัดทำเอกสารและเตรียมเสนอผลงาน							

บทที่ 2
หลักการที่เกี่ยวข้อง

2.1 หลักการที่เกี่ยวกับ Cache

2.1.1 ความหมายของ Web Caching

Web Cache จะคล้ายๆกับ Cache ของคอมพิวเตอร์ โดย Web Cache จะอยู่ระหว่าง Web Servers และ Client เครื่องเดียวหรือหลายๆเครื่อง ซึ่ง Web Cache เป็นแหล่งเก็บข้อมูลชั่วคราว จะคอยเฝ้าดูการร้องขอข้อมูล หรือที่รู้จักกันในนามของ objects เมื่อมีการร้องขอมามันจะทำสำเนาของ objects นั้นเก็บไว้ และเมื่อ Client ต้องการดึงข้อมูลจากเครือข่ายภายนอก จะมีการติดต่อไปยัง Web Cache ก่อนเพื่อตรวจสอบดูว่ามีสำเนาข้อมูลเก็บไว้ใน Web Cache หรือไม่ ถ้ามีก็จะส่งข้อมูลนั้นกลับไปยัง Client ที่ติดต่อมา ถ้าไม่มี Web Cache ก็จะทำหน้าที่ของมันโดยติดต่อไปยัง Web Site ต้นกำเนิด แล้วดึงข้อมูลมาทำสำเนาเก็บไว้ เพื่อประโยชน์ในการร้องขอครั้งต่อไป แล้วส่งต่อไปยัง Client อีกทีหนึ่ง ดังนั้นมีเหตุผลหลัก 2 ประการที่นำ Web Cache มาใช้ คือ

1. Reduce Latency คือช่วยลดระยะเวลาระหว่างการเรียกข้อมูลจนถึงเริ่มส่งข้อมูล เพราะการร้องขอข้อมูลจาก Cache จะใช้เวลาน้อยกว่าการร้องขอไปยัง Server ต้นกำเนิด
2. Reduce Traffic คือช่วยลดความคับคั่งของระบบเครือข่าย เพราะแต่ละ objects จะถูกดึงมาจาก Server เพียงครั้งเดียว เป็นการลดการสูญเสีย bandwidth

ชนิดของ Web Caching ยังแบ่งได้เป็น 2 ประเภทคือ

1. Browser Caches จะพบได้ตาม Browser ไม่ว่าจะเป็น Internet Explorer หรือ Netscape โดยมันจะทำการเก็บ objects ที่เคยเข้าไปดูมาแล้ว และเมื่อ Client กดปุ่ม "Back" เพื่อจะกลับไปยังหน้าที่เคยดูมาก่อนนี้ มันจะดึงข้อมูลจาก Browser Caches แสดงขึ้นมาอย่างรวดเร็ว
2. Proxy Caches เป็น Proxies ที่ให้บริการแก่ผู้ใช้จำนวนมากในเส้นทางเดียวกัน และเพื่อเป็นการลดจำนวนของ Internet bandwidth มันจึงทำการเก็บ objects ที่มีการเรียกใช้บ่อยๆมาไว้ที่ Proxy Caches นี้ เพื่อช่วยให้การใช้งานเร็วขึ้น

นอกจากนี้ยังมีคำที่มีความหมายใกล้เคียงหรือสัมพันธ์กับ Web Cache อยู่อีก 2 คำด้วยกัน คือ Firewall และ Gateway ในปัจจุบัน Software ส่วนใหญ่ จึงได้นำความสามารถใน

การทำงานของ Server เหล่านี้มาผสมผสานกัน ทำให้มีความสามารถและมีความยืดหยุ่นในการทำงานมากขึ้น ในองค์กรขนาดเล็กจึงอาจใช้ Web Cache ในการเป็นส่วนหนึ่งของ Proxy Firewall Server โดยให้บริหารกับเครื่องคอมพิวเตอร์ภายในองค์กร

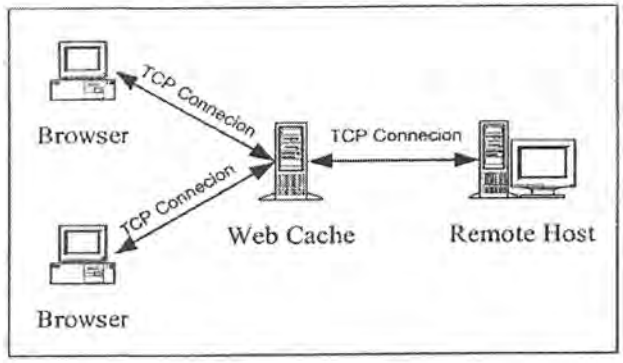
2.1.2 Simple Web Caches

2.1.2.1 การทำงานของ Simple Caching

Web browser ส่วนมากจะมีวิธีการติดต่อผ่านเครือข่ายอย่างง่ายๆ โดยระบุ URL ซึ่งประกอบไปด้วยชื่อ Host และ Item ที่ต้องการซึ่งอยู่บน Host นั้น มันจะสร้างการติดต่อแบบ TCP ไปยัง Host ที่ระบุนั้นและนำ Item ที่ต้องการนั้นมาแสดงผล แต่ถ้าไม่พบ ผู้ใช้จะได้รับ Error message แทน เนื่องจาก Browsers มีความเป็นอิสระในการเข้าถึงข้อมูลที่ตนเองต้องการ จึงอาจมีข้อมูลเดียวกันหลายชุดซ้ำๆกันอยู่ในเครือข่าย ซึ่งเป็นการสูญเสียโดยไม่จำเป็น ดังเช่น Web Site ที่เป็นที่นิยมทั้งหลาย จะมีผู้เข้าชมในวันหนึ่งเป็นจำนวนมาก ซึ่งอาจจะผ่านเส้นทางเดียวกันในเครือข่ายสื่อสารหลายครั้ง

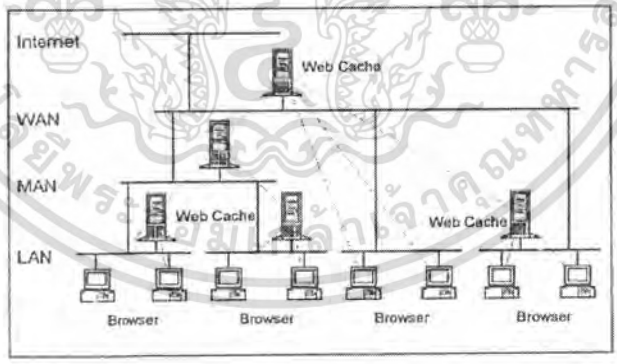
Web Caches จะช่วยลดการสูญเสียในกรณีเช่นนี้ได้ โดยการสกัดกั้นการร้องขอข้อมูลที่ซ้ำๆกัน Web Cache จะสร้าง Connection ไปยัง Host ที่อยู่ในระยะไกลเพียง Connection เดียว แล้วนำมันมาทำสำเนาส่งต่อให้กับ Browser ที่ร้องขอมาอีกทีหนึ่ง การทำเช่นนี้จะประสบผลสำเร็จก็ต่อเมื่อ Web Cache ต้องอยู่ใกล้ชิดกลับผู้ใช้งานมากที่สุดเท่าที่จะเป็นไปได้ และต้องมีผู้ใช้งานจำนวนมากพอที่จะทำให้เกิดการร้องขอข้อมูลที่มีความซ้ำกัน

การติดต่อระหว่าง Browser กับ Web Cache เหมือนกับการติดต่อระหว่าง Browser กับ Web Site ต่างกันที่การร้องขอไปยัง Web Cache ต้องส่ง URL เต็มๆไป ไม่ใช่แค่ส่วน Item ที่ต้องการ เมื่อ Web Cache ได้รับการร้องขอก็จะนำ URL นั้นไปเทียบกับการร้องขออื่นๆที่ผ่านมา หากเป็น URL ที่เคยมีการดึงข้อมูลมาจากต้นฉบับและได้มีการทำสำเนาเก็บไว้แล้ว ก็จะนำมันส่งให้กับผู้ร้องขอทันที มิเช่นนั้นแล้ว Web Cache ก็จะส่งการร้องขอต่อไปยัง Host ที่ระบุใน URL หรือ Parent Cache (จะกล่าวต่อไปในภายหลัง) แล้วนำข้อมูลที่ได้ส่งต่อให้กับ Browser อีกทีหนึ่ง ถ้าข้อมูลชุดนี้ตรงตามกฎของ Cache มันจะมีการทำสำเนาเก็บไว้ทันที เผื่อมีการร้องขอมาอีกในอนาคต การร้องขอข้อมูลของ Simple Cache ผ่านการติดต่อแบบ TCP เช่นเดียวกับ Browser สร้างการติดต่อกับ Host อื่นๆ ถ้าการติดต่อนั้นไม่สำเร็จก็จะส่งข้อความผิดพลาดให้กับผู้ใช้งาน การทำงานของ Web Cache แสดงดังรูปที่ 2.1



รูปที่ 2.1 การทำงานของ Web Cache

Cache สามารถส่งต่อการร้องขอไปยัง Cache อื่นๆได้ ซึ่งเป็นโครงสร้างแบบ Hierarchy แบ่งได้เป็นหลายระดับ ให้บริการโดยอ้อมกับผู้ใช้งานจำนวนมากมาย แต่อย่างไรก็ตามมีการกำหนดว่าแต่ละ Client (ไม่ว่าจะเป็น Browser หรือ Cache เองก็ตาม) สามารถมี Parent Cache ได้จำกัดเพียงหนึ่งเท่านั้น โครงสร้าง Hierarchy จึงมีลักษณะเหมือน Simple Tree แสดงดังรูปที่2.2 โดยมี Browser อยู่ที่จุดล่างสุดต่อจาก Local Cache และการร้องขอข้อมูลต้องทำผ่านขึ้นไปยังจุดสูงสุด และต่อไปยัง Host ที่เป็นต้นฉบับ อย่างไรก็ตาม Browser ก็สามารถติดต่อโดยตรงกับ Cache ที่ระดับบนๆได้ โดยไม่จำเป็นต้องผ่าน Local Cache



รูปที่2.2 โครงสร้าง Cache Hierarchy

2.1.2.2 ปัญหาของ Simple Caching

Cache Hierarchy นี้สามารถลดความคับคั่งของเครือข่ายได้ แต่จะเป็นปัญหาเรื่องการเก็บข้อมูลใน Disk ที่ซ้ำๆกัน ในแต่ละ Cache เพราะเมื่อมีการร้องขอผ่านในแต่ละชั้นของ Cache Hierarchy ข้อมูลที่ได้จะถูกเก็บไว้ในแต่ละ Cache ที่มันผ่าน อีก

ปัญหาที่สำคัญของ Cache Hierarchy ก็คือ Cache ในระดับเดียวกันไม่สามารถติดต่อถึงกันได้ ซึ่งอาจเป็นไปได้ที่จะมีข้อมูลอยู่ที่พื้นที่ข้างเคียงอยู่แล้ว ไม่จำเป็นต้องไปขอข้อมูลจาก Parent Cache ซึ่งมีค่าใช้จ่ายสูง และมีความคับคั่งของเครือข่ายมากอยู่แล้วเสมอไป

Cache มีพื้นที่ Disk เก็บข้อมูลอยู่จำกัด ดังนั้นจึงต้องลบข้อมูลเก่าทิ้งอยู่เสมอเพื่อให้มีพื้นที่เหลือว่างพอสำหรับการร้องขอข้อมูลใหม่ โดยส่วนใหญ่ใช้หลักการลบข้อมูลที่มีการร้องขอมาน้อยที่สุด (LRU หรือ Least Recently Uses) และในอนาคตมีโอกาสใช้มันน้อย ถ้าเรามีพื้นที่ Disk ไม่เพียงพอ จะทำให้ Cache ต้องลบข้อมูลออกจำนวนมาก หากมีการร้องขอมาใหม่ Cache จะต้องสร้าง Connection และนำข้อมูลมาอีกครั้งโดยไม่จำเป็น จำนวนพื้นที่ Disk ที่ต้องการขึ้นอยู่กับจำนวนผู้ใช้งานที่มันรองรับและสิ่งที่ผู้ใช้งานอ่านมา ในทางอุดมคติ Cache ควรจะมีพื้นที่เพียงพอสำหรับเก็บข้อมูลทุกๆข้อมูล ที่ผู้ใช้งานร้องขอมามากกว่า 1 ครั้งตลอดช่วงอายุของข้อมูล แต่ในความเป็นจริงแล้ว เราต้องเก็บข้อมูลไว้ทั้งหมดเพราะไม่สามารถจะทำนายได้ว่าในอนาคตข้อมูลใดจะถูกอ่านอีกครั้ง ความสัมพันธ์ระหว่างจำนวนผู้ใช้งานและพื้นที่เก็บข้อมูลแสดงให้เห็นว่า Cache ในระดับบนของ Hierarchies มีความต้องการพื้นที่ Disk เก็บข้อมูลมากกว่า Cache ในระดับล่าง

สำหรับการทำงานของ Cache เครื่องหนึ่ง สามารถที่จัดการข้อมูลและรองรับการร้องขอจาก Browser ที่มันให้บริการอย่างเสมอภาค แต่ถ้ามองในแต่ละ Cache จะต้องมีพื้นที่ Disk เพียงพอสำหรับเก็บข้อมูลที่ผู้ใช้งานร้องขอทั้งหมด ถ้าไม่มีข้อมูลหรือข้อมูลได้ถูกลบไปแล้ว การร้องขอจะต้องทำต่อไปยัง Parent Cache หรือ Host ที่เป็นต้นฉบับเท่านั้น ถึงแม้ว่าข้อมูลนั้นจะมีอยู่ใน Cache เครื่องอื่นๆ ที่อยู่ในระดับเดียวกันก็ตาม

2.1.3 Cache ที่ทำงานประสานกัน (Co-operating Cache)

เป็นการทำงานประสานร่วมกันระหว่าง Browser และ Cache หรือแม้แต่ Cache ด้วยกันเอง

2.1.3.1 ความชาญฉลาดของ Client

Browser หรือ Cache โดยส่วนใหญ่จะมีรายการติดต่อที่แน่นอนอยู่แล้วว่าควรจะติดต่อโดยตรงกับ Cache เครื่องใด ซึ่งการติดต่อโดยตรงจะรวดเร็ว และไม่เสียเนื้อที่เก็บข้อมูลใน Cache การเลือกใช้ Cache โดยพิจารณาจาก URL เป็นหลัก จะช่วยให้สามารถเลือกใช้ Cache ที่มีความพิเศษในแต่ละลักษณะของ URL ได้ และ

สิ่งนี้จะช่วยลดการซ้ำของข้อมูลในแต่ละ Cache ด้วยในอีกทางหนึ่ง ถ้า Cache มีปัญหาและหยุดรับการร้องขอ Client จะพยายามค้นหาเส้นทางอื่นที่จะไปถึง Host ต้นฉบับแทน อย่างไรก็ตามมี Browser น้อยมากที่มีลักษณะยืดหยุ่นเช่นนี้ อาจเป็นเพราะ Cache จะถูกกำหนดให้ทำหน้าที่เป็น Firewall ป้องกันการติดต่อโดยตรง ใน Auto-configuration script ยอมให้มีการกำหนดรายการ Cache ได้หลายเครื่องตามลำดับ โดยกำหนดเป็นชื่อ IP Address ก็ได้ และหากทั้งหมดไม่มีการตอบรับก็สามารถติดต่อโดยตรง กลไกนี้กลายมาเป็นมาตรฐานของ Internet Application อื่นๆ โดยปริยาย อย่างไรก็ตาม Browser โดยส่วนใหญ่ก็จะเลือก Cache ในลำดับแรกๆ บน DNS listแท่านั้น ซึ่ง DNS Server นี้จะรวม Cache Server ไว้เป็นกลุ่มตามลำดับ เพื่อจะแบ่งการทำงานอย่างคร่าวๆ

ในการติดต่อกับ Cache เพื่อตรวจสอบว่า Cache ยังคงให้บริการอยู่ จะใช้การติต่อแบบ TCP เพื่อร้องขอข้อมูล ถ้าการติดต่อไปยัง Cache เครื่องแรกถูกปฏิเสธ Browser จะพยายามติดต่อกับ Cache เครื่องถัดไปบน list นั้น ส่วนในอีกทางเลือกหนึ่งการติดต่อกับ Cache ในครั้งแรกอาจใช้ Packet แบบ UDP และสร้างการติดต่อแบบ TCP เมื่อมีการตอบรับ Packet แบบ UDP อาจถูกส่งไปยัง Port พิเศษ ซึ่งกำหนดไว้โดย Cache Software หรือส่งไปยัง Echo Port ซึ่งเป็นส่วนหนึ่งของมาตรฐาน TCP/IP การใช้ Echo Port จะเป็นการตรวจสอบอย่างง่ายๆ ว่า Server ยังคงอยู่แต่จะไม่บ่งบอกแน่ชัดในแง่ที่ว่า Cache Software ยังคงทำงาน การตรวจสอบแบบนี้ดูจะเป็นวิธีที่ด้อยกว่าวิธีการติดต่อแบบ TCP เสียอีกเพราะเป็นการเพิ่ม Packet การติดต่อในกรณีที่ Cache ยังคงทำงานอยู่อีกด้วย อย่างไรก็ตามถ้าหาก Cache หลายๆเครื่องยังคงทำงานอยู่ การติดต่อด้วย Packet UDP ในตอนเริ่มต้นสามารถใช้ในการหาว่าเครื่องใดให้การตอบรับเร็วที่สุด โดย Packet จะถูกส่งออกไปพร้อมๆกันและจะวัดเวลาในการตอบสนอง (Round-trip Time) Cache เครื่องใดมีการตอบสนองเร็วที่สุดจะถูกเลือกให้รับการร้องขอข้อมูลโดยใช้การติดต่อแบบ TCP เวลาในการตอบสนองของ Cache จะขึ้นอยู่กับความเร็วและภาระงานของแต่ละ Cache รวมไปถึงผลจากระยะทางและความคับคั่งของเครือข่ายด้วย

การส่ง Packet แบบ UDP ก่อนการสร้างการติดต่อแบบ TCP ดูเหมือนจะเป็นการสูญเสีย Network Bandwidth แต่ในความเป็นจริงแล้วการแลกเปลี่ยนข้อมูลแบบ UDP ต้องการเพียงแค่ 2 Packet ในแต่ละครั้งของการติดต่อเท่านั้น ซึ่งถ้าเทียบกับการแลกเปลี่ยนข้อมูลแบบ TCP ต้องมี Packet ถึง 8 Packet เป็นอย่างน้อย เราสามารถเพิ่มข้อมูลการร้องขอลงไปใน UDP Packet โดยไม่เป็นการเพิ่มความคับคั่งของเครือข่ายได้ ส่วนการตอบรับด้วย Packet UDP จาก Cache ไม่เพียงแต่ระบุ

สถานะและความเร็วเท่านั้น แต่ยังสามารถบรรจุข้อมูลที่ Client ต้องการได้ ถ้ามีข้อมูลนั้นอยู่ใน Cache และข้อมูลนั้นเล็กพอที่จะบรรจุอยู่ใน UDP Packet เพียง Packet เดียว ดังนั้นจึงสามารถลดขั้นตอนการสร้าง TCP Connection ได้ทั้งหมด

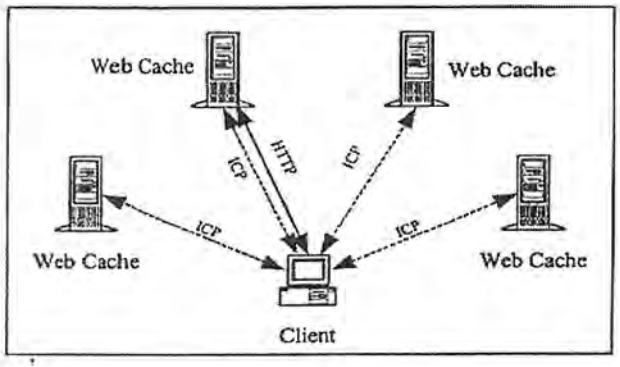
โดยสรุปการใช้ความชาญฉลาดของ Client สามารถแบ่งได้เป็น 5 ระดับ คือ

ตารางที่ 2.1 สรุปการใช้ความชาญฉลาดของ Client

0	Simple	ใช้ Cache เพียงเครื่องเดียว แต่สามารถติดต่อโดยตรงได้ในบางรูปแบบของ URL
1	Selection	เลือกใช้ Cache ได้หรือ ติดต่อโดยตรงโดยดูจากรูปแบบของ URL เป็นหลัก
2	Resilience	มีการพยายามติดต่อกับ Cache เครื่องอื่นๆ เมื่อตรวจสอบแล้วพบว่า Cache เครื่องแรกไม่ทำงาน
3	Load Balancing	เลือกใช้ Cache โดยดูจากความเร็วในการตอบสนอง
4	Discovery	เลือกใช้ Cache ซึ่งมีข้อมูลที่ Client ต้องการอยู่ด้วย

2.1.3.2 การทำงานของ Co-operating Cache

ใน Harvest Project ที่มหาวิทยาลัย Colorado สหรัฐอเมริกา ได้ทำการพัฒนา Internet Cache Protocal (ICP) เพื่อจุดมุ่งหมายในการทำ Load Balancing และ Discovery ซึ่งปัจจุบัน ICP ได้กลายมาเป็นมาตรฐานของ Internet สำหรับการติดต่อระหว่าง Cache ICP ใช้การส่ง Packet UDP เพื่อการร้องขอข้อมูลเริ่มต้นเท่านั้น แต่ถ้าต้องการส่งข้อมูลในแบบ TCP จะใช้ Hypertext Transfer Protocol (HTTP) ธรรมดา การติดต่อระหว่าง ICP Cache แสดงดังรูปที่2.1.3.1 ซึ่งการติดต่อแบบ ICP โดยใช้ UDP Packet แสดงด้วยเส้นประ ขณะที่การติดต่อแบบ HTTP โดยใช้ TCP Connection แสดงด้วยเส้นทึบ



รูปที่2.3 รูปแบบการติดต่อของ Co-operating Cache

แต่ละ Cache จะมีรายการของ Cache เครื่องอื่นๆ ซึ่งรูปแบบอาจกำหนดให้เลือกแต่ละ Cache โดยดูจาก URL เป็นหลัก เมื่อได้รับการร้องขอข้อมูล ซึ่งตนเองไม่มีข้อมูลนั้นอยู่ มันจะส่ง ICP Packet ไปยังเครื่องที่เหมาะสมพร้อมๆกัน แล้วรอการตอบรับกลับเพื่อป้องกันกรณีที่ Packet มีการสูญหาย หรือ Cache ที่ส่งไปนั้นไม่ทำงาน จะมีการกำหนดเวลาที่จะต้องรอไว้ ถ้า Cache ได้รับการตอบรับว่า 'hit' จะหมายความว่ามีข้อมูลที่ร้องขอไป ดังนั้นมันจะดึงข้อมูลนั้นโดยการสร้าง HTTP Connection ไปยัง Server ที่ตอบรับ 'hit' ที่เร็วที่สุด แต่ถ้าการตอบรับมีเพียงแต่ 'miss' เท่านั้น Client จะพิจารณาเลือก Cache ที่ถูกกำหนดไว้ให้เป็น 'parent' เท่านั้น (ไม่ใช่ 'neighbor') โดยจะสร้าง HTTP Connection ไปยัง Parent Cache ที่มีการตอบรับ 'miss' ที่เร็วที่สุด แม้ในขณะนี้ Client จะรู้อยู่แล้วว่า Parent Cache จะต้องส่งต่อการร้องขอต่อไป แต่อย่างน้อยก็เป็นวิธีการที่ดีที่สุดแล้ว ICP ถือว่าเวลาที่ตรวจจับในการตอบรับต้องมีความสัมพันธ์กัน แต่ถ้ามีเหตุผลในทางเทคนิคที่ต้องการให้เลือกใช้ Cache บางเครื่อง ถึงแม้ว่าจะมีเวลาในการตอบรับช้าก็ตาม สามารถนำ Weighting Factor มาประยุกต์ใช้ได้

Hierarchy ของ Parent และ Neighbor ที่นำมาใช้ใน ICP มีความเป็นอิสระมากกว่าการเป็นโครงสร้างแบบ Rigid Tree แต่จะเป็นไปตามลักษณะเฉพาะของแต่ละ Cache ในช่วงของการติดต่อเพื่อค้นหาข้อมูลจะไม่มีความแตกต่างระหว่าง Cache ทั้ง 2 ชนิด จะต่างกันก็เพียงแต่ว่า การค้นหาข้อมูลที่ไม่มีอยู่ใน Cache ใดๆ เลย Client จะคิดว่า Parent ของมันจะอยู่ใกล้กับแหล่งที่มาของข้อมูลมากกว่า เราสามารถกำหนดให้ Cache ทำสำเนาข้อมูลซึ่งส่งมาจาก Cache เครื่องอื่นๆหรือเพียงแต่ส่งมันต่อไปยัง Client ของมันก็ได้ ซึ่งกรณีการส่งต่อจะเป็นการลดการใช้พื้นที่ Disk ในแต่ละ Cache แต่จะเป็นการเพิ่มความคับคั่งของเครือข่ายระหว่าง Cache

ด้วยกัน Cache Software โดยส่วนใหญ่สามารถเลือกได้โดยอิสระสำหรับแต่ละ Neighbor หรือ Parent ว่าจะให้ทำการสำเนาข้อมูลหรือเพียงแต่ส่งต่อเท่านั้น ดังนั้นจึงสามารถปรับความสมดุลระหว่างการใช้พื้นที่ Disk และความคับคั่งของเครือข่ายได้

จะพบว่า Client ที่ทำการร้องขอในแบบ ICP มักจะเป็น Cache ด้วยกัน ซึ่งไม่มีเหตุผลที่แน่นอนว่าทำไมจึงไม่ใช้ Protocol นี้ติดต่อกันระหว่าง Browser ด้วยกัน อาจเป็นเพราะความยากในการจัดการ Browser ที่ต่างยี่ห้อและต่างรุ่นกันจำนวนมากมาย และโดยทั่วไปแล้วจะมี Local Cache เพียงหนึ่งเครื่องที่คอยรับข้อมูลจากภายนอกอยู่แล้ว ดังนั้นจึงไม่จำเป็นต้องมีความสามารถเต็มที่เช่นเดียวกับ Co-operating Cache

2.1.3.3 การทำงานร่วมกับ Simple Cache

จากส่วนที่ผ่านมาได้อธิบายถึงการทำงานระหว่าง Co-operating Cache ด้วยกันเท่านั้น แต่ในทางปฏิบัติอาจต้องกำหนดให้ทำงานร่วมกับ Simple Cache ดังนั้นจึงต้องนำหลักการทำงานมาพิจารณาร่วมกัน Simple Cache สามารถอ้างโดย ICP Client ได้ แต่โดยปรกติจะถูกกำหนดให้เป็น Parent เท่านั้น เพราะว่า Simple Cache ไม่สามารถรองรับการติดต่อแบบ ICP จะทำได้ก็แต่เพียงการติดต่อไปยัง TCP/IP Echo Port เท่านั้น ถ้า Simple Cache ยังคงทำงานอยู่ก็จะมี Packet ตอบกลับมาเช่นกัน ซึ่งจะถูกแปลความหมายเช่นเดียวกับ 'miss' เสมอ ดังนั้นหากกำหนดให้ Simple Cache เป็น 'neighbor' ก็จะไม่มีการดึงข้อมูลมาจากเครื่องนั้น และถ้าไม่มีการตอบรับ 'hit' จากเครื่องอื่นๆ อาจเป็นไปได้ที่ว่า Simple Cache จะถูกเลือกให้เป็น Parent Cache ที่ตอบรับ 'miss' ที่เร็วที่สุด ถ้า ICP Client มีทั้ง Simple Cache และ Co-operating Cache ที่เป็น Parent จะมีความเป็นไปได้สูงที่ Simple Cache จะถูกเลือกก่อนเสมอ เพราะมันมีงานที่จะต้องทำน้อยกว่าก่อนที่จะตอบรับ เพื่อป้องกันปัญหานี้ อาจมีการนำระบบ weighting มาใช้ ให้การเลือกเอนเอียงไปทาง Co-operating Cache มากกว่า แม้ว่า Simple Cache จะมีการตอบรับก่อนก็ตาม ในทางกลับกัน หากมีการร้องขอข้อมูลจาก Simple Cache ไป Co-operating Cache การทำงานจะเป็นไปตามปรกติ ซึ่งจะคล้ายกับการที่ Browser ติดต่อไปยัง Co-operating Cache

2.1.3.4 ผลกระทบที่เกิดขึ้น

การทำงานของ Co-operating Cache จะไม่มีผลกระทบต่อระบบเครือข่าย เมื่อข้อมูลถูกพบใน Cache ของ Browser ก่อน หรือคิดว่า ICP Timeout มีค่าน้อยมากเมื่อเทียบกับเวลาที่ใช้ในการดึงข้อมูลมาจากเครื่องต้นฉบับ ซึ่งข้อมูลไม่ได้มีการเก็บไว้ใน Cache เครื่องใดๆที่อยู่ระหว่าง Browser ถึงเครื่องต้นฉบับ การใช้ Co-operating Cache ที่ระดับใดระดับหนึ่งของ Hierarchy จะเป็นการเพิ่มจำนวนของ Cache ที่ Browser จะต้องติดต่อในทางอ้อม ดังนั้นจึงเป็นการเพิ่มโอกาสในการค้นพบข้อมูลมากขึ้นสำหรับข้อมูลซึ่งมีขนาดไม่เล็กพอที่จะบรรจุอยู่ใน Packet แบบ ICP ที่ตอบรับกลับ จำนวนของความคับคั่งในเครือข่ายจะเพิ่มขึ้นเล็กน้อยจากการใช้ Packet พิเศษซึ่งใช้ UDP Protocol แต่สำหรับข้อมูลขนาดเล็กเป็นไปได้ที่ความคับคั่งของเครือข่ายอาจจะลดลง เนื่องจากไม่มีความจำเป็นต้องสร้าง TCP Connection ในการส่งข้อมูล

Co-operating Cache ซึ่งต่ออยู่ในเครือข่ายความเร็วสูง สามารถที่จะกำหนดให้มีการแบ่งปันพื้นที่ Disk ซึ่งเป็นการลดความต้องการในการใช้ Disk ขนาดใหญ่ในแต่ละเครื่อง แต่จะเป็นการเพิ่มความคับคั่งของเครือข่ายระหว่าง Cache ความสามารถในการทำงานของ Co-operating Cache มีความยืดหยุ่นและสามารถแบ่งเบาภาระได้ ซึ่งจะเป็นผลดีกับผู้ใช้งาน

2.1.3.5 ความถูกต้องของข้อมูลใน Cache

เมื่อไรก็ตามที่มีการทำสำเนาจากข้อมูลต้นฉบับก็จะมีปัญหาในการจัดการเกิดขึ้น ถ้าข้อมูลต้นฉบับมีการเปลี่ยนแปลงหลังจากที่ได้มีการทำสำเนาเก็บไว้ มันจะกลายเป็นข้อมูลที่พ้นสมัย(out of date) ในทันที เรียกว่าปัญหา Consistency ความล้าสมัยของสำเนาข้อมูลถูกกำหนดเป็นช่วงเวลาตั้งแต่ข้อมูลต้นฉบับแปลี่ยนไป และมักจะแปรผันไปตามสัดส่วนของอายุของข้อมูลมันเป็นไปได้ยากที่จะติดต่อกับที่มาว่าสำเนาข้อมูลนี้ล้าสมัยแล้วหรือยัง ถ้าให้ระบบเครือข่าย และ Host ที่ให้บริการไม่มีการเปลี่ยนแปลงใดๆ เนื่องจาก Hypertext Transfer Protocol ในรุ่นก่อนๆ มีเพียงแต่การร้องขอแบบ 'GET' เท่านั้น ซึ่งจะได้รับข้อมูลมาทั้งหมด เหมือนกับขบวนการร้องขอข้อมูลแล้วนำมาเปรียบเทียบกับสำเนา

การปรับปรุงสิ่งแรกก็คือเพิ่มการร้องขอแบบ 'HEAD' ซึ่งจะมีการส่งข้อมูลสรุปเกี่ยวกับ ข้อมูลที่ร้องขอ เช่นเวลาที่เปลี่ยนแปลงครั้งหลังสุด ถ้าข้อมูลนั้นแสดงให้เห็นว่าสำเนามีความล้าสมัย มันจะต้องสร้าง Connection ไปยังเครื่องต้นฉบับเพิ่ม

ขึ้นอีกเพื่อร้องขอข้อมูลแบบ 'GET' ข้อด้อยนี้ทำให้มีการนำการร้องขอในแบบมีเงื่อนไขมาใช้ ซึ่งเรียกกันว่า 'If-Modified-Since GET' หรือ 'IMS GET' การร้องขอแบบนี้จะมีการนำเวลาของสำเนาข้อมูลมาใส่ไว้ในการร้องขอด้วย แล้วคาดหวังว่าการตอบกลับจะเป็นเพียง 'unchanged' หรือการส่งข้อมูลมาใหม่ ดังนั้นทั้งการตรวจสอบและการแก้ไขข้อมูลสามารถทำได้ในการติดต่อเพียง Connection เดียวเท่านั้น ปัจจุบัน Web Server โดยส่วนใหญ่สนับสนุนการร้องขอแบบ 'IMS GET' แต่การตรวจสอบแบบนี้ยังคงต้องมีการติดต่อไปยัง Host ที่เป็นต้นฉบับ ดังนั้นจึงอาจจะต้องใช้เวลาบ้างสำหรับ Site ที่อยู่ในระยะไกล แต่มันจะช่วยลดปริมาณความต้องการใช้ระบบเครือข่ายเมื่อข้อมูลไม่มีการเปลี่ยนแปลง

แต่อย่างไรก็ตามมันยังไม่เป็นที่น่าพอใจ โดยเฉพาะข้อมูลขนาดเล็กที่อยู่ระยะไกล การสร้าง Connection ไปยังเครื่องต้นฉบับ อาจใช้เวลามากกว่าการส่งข้อมูลเสียอีก ถึงแม้ว่าจะมีการตรวจสอบความล้าสมัยด้วยการร้องขอแบบมีเงื่อนไข (IMS GET) มันก็ยังใช้เวลาเกือบจะเท่ากับการที่ Client ร้องขอข้อมูลจากเครื่องต้นฉบับโดยตรง จากกรณีเช่นนี้จุดประสงค์ของการใช้ Cache จะสูญเสียไป ดังนั้นทางที่เป็นไปได้ ผู้ใช้งานอาจต้องมีการยอมรับข้อมูลที่มีความล้าสมัยอยู่บ้าง เพื่อความรวดเร็วในการตอบสนองในทันทีทันใดจาก Cache แต่ถ้าผู้ใช้งานรู้สึกว่าข้อมูลมีความล้าสมัยก็ยังสามารถที่จะบังคับให้ Cache ทำการดึงข้อมูลจากเครื่องต้นฉบับมาใหม่ได้ เช่นกันถ้า Cache ไม่ได้มีการตรวจสอบความล้าสมัยในทุกๆการร้องขอข้อมูล จะต้องมีวิธีการที่จะตัดสินว่า เมื่อไรจะต้องมีการตรวจสอบ วิธีที่ง่ายที่สุดคือ ตรวจสอบเมื่อช่วงระยะเวลาหนึ่งผ่านไป (ในทางปฏิบัติจะตรวจในครั้งแรกที่มีการร้องขอข้อมูลหลังจากได้เลยช่วงเวลาที่กำหนด) ดังนั้นจึงเป็นการยอมให้ผู้บริหาร Web Cache สามารถกำหนดความล้าสมัยมากที่สุดเป็นเวลาที่แน่นอน ตัวอย่างเช่น กำหนดให้ Cache ต้องไม่นำข้อมูลที่มีอายุเวลามากเกินกว่า 12 ชั่วโมงมาใช้

อีกวิธีการหนึ่งเป็นวิธีที่พยายามจะสะท้อนถึงความหลากหลายของข้อมูลบน Web ที่บ้างก็เปลี่ยนเป็นรายชั่วโมง บ้างก็ไม่เคยมีการเปลี่ยนเลย เมื่อ Cache เก็บข้อมูลมันจะกำหนดอายุเวลาของข้อมูลเอาไว้ (time to live หรือ TTL) ข้อมูลนี้จะถูกนำมาใช้โดยมิต้องมีการตรวจสอบกับต้นฉบับจนกระทั่งผ่านพ้นเวลานี้ ใน HTTP Protocol สามารถกำหนด Timestamp Headers ซึ่งนำมาใช้ในการกำหนดค่า TTL ได้ แต่โดยส่วนใหญ่ไม่ได้ถูกนำมาใช้หรือไม่มีอยู่ใน Web Server ทั้งหลาย ดังนั้นสิ่งที่นำมาใช้กันอย่างกว้างขวางก็คือเวลาที่มีการเปลี่ยนแปลงครั้งหลังสุด (Last-Modify) โดยเราจะกำหนด TTL ให้เป็นเปอร์เซ็นของอายุข้อมูลในปัจจุบัน ตัวอย่างเช่น กำหนดให้ TTL มีค่าเป็น 10% ของข้อมูลซึ่งมีการเปลี่ยนแปลงครั้งสุดท้ายเมื่อ

10 วันก่อน ดังนั้นสำเนาข้อมูลจะไม่ถูกตรวจสอบจนกว่าจะผ่าน 1 วันให้หลัง ยังมีอีกสองค่าที่โดยปรกติจะนำมาใช้ในการคำนวณหา TTL ด้วยก็คือค่ามากที่สุดที่จะปกป้องสำเนาข้อมูลให้สามารถนำมาใช้ได้โดยไม่ต้องมีการตรวจสอบ และค่า Default TTL ซึ่งจะใช้สำหรับข้อมูลที่ไม่รู้เวลาของการเปลี่ยนแปลงครั้งหลังสุด

วิธีการใช้ TTL ดูจะเป็นการเพิ่มความสามารถในการจัดการความล้าสมัยได้ในสองทางด้วยกัน คือ ข้อมูลที่ไม่มีการเปลี่ยนแปลงจะไม่ถูกตรวจสอบโดยไม่จำเป็น และข้อมูลที่มีการเปลี่ยนแปลงเสมอจะไม่ล้าสมัยจนเกินไปนัก อย่างไรก็ตามผู้ใช้งานมักจะต้องการให้มีการประกันเวลาสูงที่สุดที่ข้อมูลมีความล้าสมัยด้วย โปรแกรม Web Cache โดยส่วนใหญ่ ยอมให้มีการกำหนดค่าพารามิเตอร์ของความล้าสมัยที่ต่างกันได้ตามรูปแบบ URL ซึ่งโดยทั่วไปจะมีข้อแตกต่างระหว่างเอกสารที่เป็นข้อความและรูปภาพ เอกสารที่เป็นข้อความมักจะมีขนาดเล็กและมักมีการเปลี่ยนแปลงบ่อย ขณะที่เอกสารที่เป็นรูปภาพมีขนาดใหญ่และเปลี่ยนแปลงน้อยกว่า ความสามารถเช่นนี้สามารถนำไปใช้กับการรองรับ Protocol อื่นๆ ได้อีกด้วย เช่น FTP และ Gopher ซึ่งจะไม่ปรากฏอายุของข้อมูล

2.1.4 Cache Software

เนื่องจาก World Wide Web เป็นระบบเปิด (open System) ที่ยอมให้ระบบและโปรแกรมที่มีความแตกต่างกันสามารถทำงานร่วมกันได้ ดังนั้นจึงไม่น่าแปลกใจที่จะพบโปรแกรมจำนวนมากมายที่ทำงานได้เช่นเดียวกับ Web Cache โดยส่วนใหญ่แล้วการพัฒนาโครงสร้างพื้นฐานของระบบ Internet มักจะใช้ระบบปฏิบัติการ Unix ดังนั้นโปรแกรม Web Cache จึงมักจะทำงานอยู่บนปฏิบัติการ Unix ด้วยเช่นกัน แต่อย่างไรก็ตาม ในการทำงานกับระบบปฏิบัติการอื่นๆ ก็กำลังเป็นที่นิยมเพิ่มขึ้นอย่างรวดเร็ว ในส่วนนี้จะอธิบายถึงโปรแกรม Web Cache บางโปรแกรมซึ่งเป็นที่นิยมใช้งาน

2.1.4.1 CERN Proxy Server

โปรแกรมนี้ถูกพัฒนาขึ้นโดยใช้โปรแกรมภาษา C และนำมาใช้กับ Unix การทำงานหลักๆของโปรแกรมนี้คือการทำหน้าที่เป็น Web Server แต่มีความสามารถในการเป็น Web Cache จึงได้รับความนิยม โปรแกรมนี้ทำงานเช่นเดียวกับ Simple Cache ซึ่งสามารถมี Parent ได้เพียง 1 เครื่องเท่านั้น และรองรับโปรโตคอล HTTP, FTP, Gopher และ WAIS ได้ มันไม่สามารถตรวจสอบความล้มเหลวของ Parent และจะส่งข้อผิดพลาดถ้าไม่สามารถติดต่อได้ การตรวจสอบความล้าสมัยของข้อมูล

สามารถตรวจสอบได้ทั้งแบบกำหนดเวลาและการใช้ TTL และถ้าต้องการข้อมูลจากเครื่องที่เป็นต้นฉบับ เมื่อข้อมูลมีความล้าสมัยแล้ว จะใช้ IMS GET ในการร้องขอ

ข้อมูลทั้งหมดจะถูกเก็บเป็น File ลงใน Disk ซึ่งชื่อ File จะถูกกำหนดจาก URL เป็นผลให้โครงสร้างของข้อมูลมี Directory จำนวนมาก การเข้าถึงข้อมูลทำได้ช้า จากผลดังกล่าวทำให้ CERN Server ไม่เหมาะกับงานที่มีปริมาณมากๆ

2.1.4.2 Apache

โปรแกรมนี้ได้มีการแจกจ่ายให้ใช้ได้ฟรีจึงมีการใช้งานกันอย่างแพร่หลาย เป็นโปรแกรมที่ถูกใช้งานเป็น Web Server เป็นหลัก โปรแกรมพัฒนาด้วยโปรแกรมภาษา C และสามารถทำงานได้บน Unix เกือบทั้งหมด ในส่วนของ Caching สามารถรองรับการทำงานแบบเลือก Parent Cache โดยดูจากรูปแบบของ URL ได้ ความล้าสมัยสามารถควบคุมได้ทั้งแบบการกำหนดเวลาและการใช้เทคนิค TTL เป็นโปรแกรมที่มีความสามารถในการเป็น Web Caching เช่นเดียวกับ CERN

2.1.4.3 Netscape Proxy Server

โปรแกรมนี้เป็นโปรแกรม Web Cache และ Proxy Server โดยเฉพาะออกจำหน่ายเมื่อปี 1995 และสามารถทำงานได้กับระบบปฏิบัติการหลายๆตัว เช่น Digital Unix, HP-UX, AIX, IRIX, SunOS, Solaris, BSDI, Windows 95 และ Windows NT และสามารถควบคุมผ่านโปรแกรม Web Browser

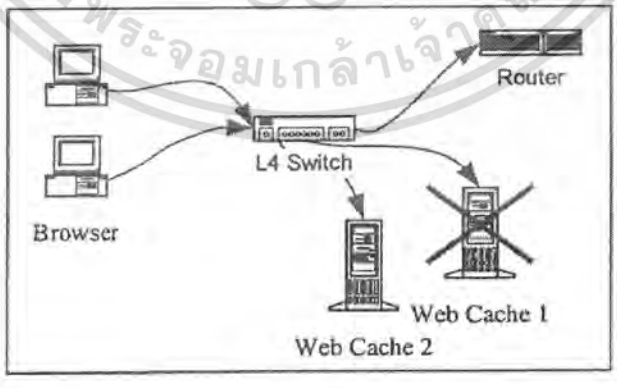
ในส่วนของ Caching สามารถรองรับการทำงานแบบเลือก Parent Cache โดยดูจากรูปแบบของ URL และสามารถตรวจสอบความล้มเหลวของ Parent Cache แล้วเลือกเครื่องต่อไปได้ และถ้าทั้งหมดไม่ทำงานจะติดต่อไปยังเครื่องที่เป็นต้นฉบับโดยตรง แต่ Netscape Proxy Server ไม่สามารถทำงานในลักษณะ Load Balancing หรือ Discovery ได้ ส่วนความล้าสมัยสามารถควบคุมได้ทั้งแบบกำหนดเวลาและการใช้เทคนิค TTL โปรแกรมนี้สามารถทำงานเป็น Firewall โดยกรองข้อมูลได้หลายรูปแบบ เช่น จากรูปแบบของ URL, บางลักษณะของ MIME types หรือข้อมูลที่มี Java, JavaScript หรือรูปภาพ ก็ได้ File ข้อมูลที่เก็บอยู่ใน Disk และมีความลึกของ Directory ไม่มากนัก ดังนั้นจึงสามารถเข้าถึงข้อมูลได้เร็ว โปรแกรมนี้สามารถทำงานร่วมกับ SNMP(Simple Network Management Protocol) โดยสามารถกำหนดเงื่อนไขซึ่งจะส่งข้อความเตือนไปยัง SNMP Server ทำให้สามารถตรวจสอบโปรแกรมจากส่วนกลางโดยผ่านอุปกรณ์ Network อื่นๆได้

2.1.4.4 Squid

Internet Research Task Force Research Group on Resource Discovery (IETF-RD) ได้มีการพัฒนาโปรแกรม Harvest เพื่อให้มีการใช้ Network ของแต่ละ Server อย่างมีประสิทธิภาพ โปรแกรมถูกออกแบบให้มีการทำงานร่วมกันเองจนกระทั่งสามารถแบ่งภาระกระจายไป ระหว่าง Server หลายๆตัวเป็นที่มาของ Internet Cache Protocol (ICP) สำหรับการติดต่อระหว่างServer โปรแกรมสามารถใช้กับ SunOS และ Digital Unix การ Configuration ทำได้โดยการแก้ไขText File ซึ่งจะถูกอ่านเมื่อโปรแกรมเริ่มทำงาน โปรแกรมสามารถรองรับโปรโตคอล HTTP, FTP และ Gopher ได้ ค่า Default และ Maximum TTL สามารถกำหนดให้มีค่าต่างกันได้สำหรับรูปแบบ URL ที่ต่างกัน แต่เมื่อข้อมูลมีความล้าสมัยแล้วก็จะถูกลบไป โดยไม่มีการตรวจสอบโดย IMS GET

โปรแกรม Harvest ถูกนำมาใช้ในโครงการ NLANR Cache และได้ถูกเปลี่ยนชื่อมาเป็น Squid มีการเปิดเผย Source Code แจกจ่าย เพื่อให้อาสาสมัครได้มีการทดสอบและปรับปรุงโปรแกรม ได้มีการเพิ่มความสามารถในการรองรับ Connection ซึ่งมีการใช้โปรโตคอล HTTPS และ SSL

โปรแกรม Squid มีความสามารถแบ่งปันเนื้อที่ Disk โดยไม่เก็บข้อมูลจาก Cache ที่ระบุได้ สามารถกำหนด Weight ให้กับ Parent Cache และสามารถใช้ IMS GET เพื่อตรวจสอบความล้าสมัยจากข้อมูลที่หมดอายุได้ และยิ่งไปกว่านั้นสามารถควบคุมการเข้าใช้ โดยดูจาก Source และ Destination Address. HTTP method, โปรโตคอล, โดเมน, Port, เวลาและรูปแบบของ URL



รูปที่2.4 การใช้ Layer 4 Switch กับ Web Cache

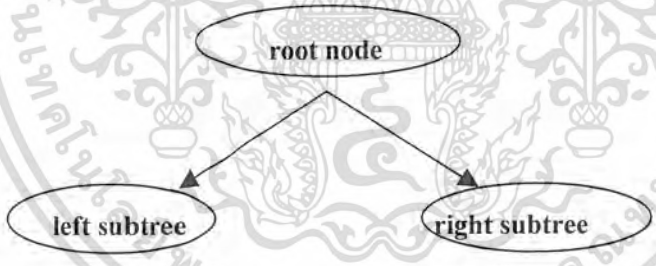
2.2 ไบนารีทรี(Binary Tree)

โครงสร้างของข้อมูลในระบบคอมพิวเตอร์ที่มีลักษณะคล้ายกับต้นไม้ซึ่งแต่ละจุดมีความสัมพันธ์กันในกิ่งก้านสาขา โดยที่จุดแตกกิ่งก้านเราเรียกว่า โหนด(node) ซึ่งเราสมมติให้เป็นตัวเก็บข่าวสารไว้ โครงสร้างแบบทรีนี้จะเป็นโครงสร้างข้อมูลไม่เชิงเส้น(non-linear)แบบที่สำคัญมาก เพราะจะเป็นพื้นฐานในการจัดการข้อมูลที่มีความสัมพันธ์กัน

2.2.1 นิยามและหลักการ

ในหัวข้อนี้เราจะกล่าวถึงแต่โครงสร้างข้อมูลที่เป็นแบบ ไดเร็กทรี(direct tree) ซึ่งมีคุณสมบัติคือมีโหนดอยู่ที่จุดยอด 1 โหนด ซึ่งเรียกว่า รูตโหนด(root node)ซึ่งมีส่วนที่เรียกว่าต้นไม้ย่อย(subtree) และมีส่วนที่เป็นใบ(leave node)คือจะมีจำนวนต้นไม้ย่อยเป็น 0 หรือบางครั้งเรียกว่า outdegree หรือ degree 0 ฉะนั้นอาจกล่าวได้ว่าดีกรีก็คือจำนวนต้นไม้ย่อยนั่นเอง ถ้ามีโหนดที่มีดีกรีไม่เท่ากับ 0 เรียกว่า non-terminal node

ไบนารีทรีจะประกอบไปด้วยรูตโหนด(root node) ต้นไม้ย่อยที่ทางซ้าย(left subtree) และต้นไม้ย่อยทางขวา(right subtree)ไบนารีทรีที่มีต้นไม้ย่อยทางซ้าย(left subtree)เท่ากับต้นไม้ย่อยทางขวา(right subtree)จะเรียกว่า ไบนารีทรีที่สมบูรณ์(complete binary tree)และไบนารีทรีที่มีรูตโหนดเพียงโหนดเดียว จะเรียกว่า ไบนารีทรีที่ว่างเปล่า(empty binary tree)



รูปที่ 2.5 แสดงตัวอย่างของไบนารีทรี

2.2.2 การกระทำบนไบนารีทรี

ในหัวข้อที่แล้วเราได้กล่าวถึงโครงสร้างของไบนารีทรีไปบ้างแล้ว ส่วนหัวข้อนี้ เราจะกล่าวถึงการกระทำในไบนารีทรี เช่น การท่องไปในทรี(traversal of tree) การเพิ่ม (insertion) การลบออก (deletion) การค้นหา(searching) และ การก๊อปปี้ (copying)

2.2.2.1 การท่องไปในทรี

การท่องไปในทรี(traverse)ในที่นี้จะหมายถึง การเดินทางในทรีโดยจะต้องผ่านโหนดทุกโหนดและจะต้องกระทำกับโหนดอย่างน้อย 1 ครั้ง คำว่ากระทำ (process) ในที่นี้คือการประมวลผลบางอย่าง เช่น การค้นหาเป็นต้น เราจะกล่าวถึงเฉพาะการท่องไปในทรีของไบนารีทรีเท่านั้น ซึ่งมีวิธีเดินที่สำคัญอยู่ 3 วิธี คือ

2.2.2.1.1 การท่องทรีแบบพรีออร์เดอร์(preorder traversal)

2.2.2.1.2 การท่องทรีแบบอินออร์เดอร์(inorder traversal)

2.2.2.1.3 การท่องทรีแบบโพสต์ออร์เดอร์(postorder traversal)

2.3 การวิเคราะห์อัลกอริทึม

ในการเขียนโปรแกรมคอมพิวเตอร์เพื่อแก้ไขปัญหาต่างๆนั้น มักจะมีขั้นตอนเริ่มจากการวิเคราะห์ลักษณะปัญหา หาทางแก้ นำมาเขียนเป็นโปรแกรม ทดสอบและแก้ไข จากที่กล่าวมา การที่เราพยายามแก้ไขปัญหาใดๆนั้นก่อให้เกิด *อัลกอริทึม* หรือกระบวนการแก้ปัญหา ซึ่งจะอยู่ในรูปของลำดับคำสั่งที่มีเป้าหมายชัดเจน

2.3.1 การวิเคราะห์การทำงานของโปรแกรม

ในหัวข้อนี้ จะพูดถึงการทำงานของโปรแกรมโดยเน้นให้ถึงเวลาที่ใช้ในการทำงานของโปรแกรม ในการแก้ปัญหาต่างๆสิ่งที่เป็นปัญหาก็คือ หลักการที่จะเลือกใช้อัลกอริทึม ซึ่งในที่นี้จะกล่าวถึง2 วิธีที่ขัดแย้งกันก็คือ

1. ต้องการ อัลกอริทึม ที่สามารถเขียนโค้ด ดีบัก (debug) และทำความเข้าใจ อัลกอริทึมได้ง่าย
2. ต้องการอัลกอริทึมที่ใช้ประโยชน์เครื่องคอมพิวเตอร์นั้นสูงสุด โดยเฉพาะอย่างยิ่งความรวดเร็วในการทำงาน

ซึ่งจะเห็นว่าทั้งสองข้อนี้มักเดินสวนทางกันเสมอ คือ ถ้าต้องการให้โปรแกรมมีการทำงานที่รวดเร็ว โปรแกรมนั้นก็มักทำความเข้าใจยากและซับซ้อน แต่ถ้าต้องการโปรแกรมที่เข้าใจง่าย ก็มักจะเป็นโปรแกรมที่ยาว และมักจะมีการทำงานที่ช้า ดังนั้นเราควรหาทางที่จะให้มีความสมดุลของทั้งสองกรณีให้มากที่สุด

2.3.2 การวัดความเร็วของโปรแกรม

ความเร็วของโปรแกรมมักขึ้นอยู่กับปัจจัยต่างๆดังนี้

1.อินพุต(input)ของโปรแกรม

2.คุณภาพของโค้ดที่ได้จากตัวคอมไพเลอร์ในการสร้างโปรแกรม

3.ความเร็วของคำสั่งในตัวเครื่องที่จะใช้งาน

4.โครงสร้างที่ใช้เขียนโปรแกรมนั้น

เป็นความจริงว่าความเร็วในการทำงานจะขึ้นกับขนาดของอินพุตที่เข้ามา โดยถือว่าปัจจัยข้อที่ 2 , 3 และ 4 คงที่หรือพูดได้อีกอย่างว่าเวลาการทำงานเป็นฟังก์ชั่นของอินพุต ตัวอย่างที่จะชี้ให้เห็นในเรื่องนี้คือเรื่องของการเรียงลำดับข้อมูลและ จะใช้ ฟังก์ชั่น T(n) เพื่อบอกถึงเวลาในการทำงานของโปรแกรมนั้นว่าเป็นฟังก์ชั่นอะไรเทียบกับอินพุต (n)ก่อนอื่นจะขออธิบายถึงตัวอย่างอัตราการเพิ่มของฟังก์ชั่นที่ใช้เสียก่อน เรามักเคยเห็นสัญลักษณ์ $\mathbf{O}(n^2)$ ซึ่งจะหมายถึง การแสดงค่าการเพิ่มขอเวลาที่โปรแกรมนั้นใช้ในขณะที่มีอินพุตเพิ่มเป็นจำนวน n เครื่องหมาย $\mathbf{O}(n^2)$อ่านว่า "Big Oh of n^2 "หมายความว่า เวลา ในการทำงานจะเป็นสัดส่วนกับจำนวนอินพุตเป็นค่ายกกำลังสอง ดังนั้นเมื่อเราทราบว่าเวลาในการทำงานของโปรแกรมเป็นสัดส่วนกับอินพุต ในสภาวะแวดล้อมเดียวกันดังนั้นเราควรเลือกใช้อัลกอริทึมที่ให้ค่าเวลาการทำงานมีอัตราการเติบโตน้อยที่สุดเมื่ออินพุตเพิ่มขึ้น

สรุป กฎในการคำนวณหาเวลาในการทำงานของโปรแกรม

ต่อไปนี้จะเป็นการสรุปกฎต่างๆ ที่ใช้ในการหาประสิทธิภาพของโปรแกรมโดยมีอินพุตเป็นจำนวน n

1. ในการทำงานที่เกี่ยวกับกระบวนความ "เท่ากับ", "เขียนข้อมูล" ,"อ่านข้อมูล"จะใช้เวลา $\mathbf{O}(1)$
2. เวลาที่ใช้ในกลุ่มกระบวนความที่เรียงตามลำดับกันจะมีค่าเป็นผลรวมของกระบวนความใดๆในกลุ่มที่เรียงลำดับนั้นใช้มากที่สุด
3. เวลาที่ใช้กับกระบวนความ if จะขึ้นกับการทำงานในส่วนเปรียบเทียบของกระบวนความบวกกับเวลาที่ใช้ในการทำงานในส่วนของเงื่อนไข

4. ในการทำงานที่เป็นลูปจะพิจารณาผลรวมการทำงานในแต่ละรอบการทำงานที่ผ่านไปนั่นคือ ผลรวมในการทำงานจะเป็นค่าคงที่ค่าหนึ่งที่ใช้ในลูปนั้น คูณกับเวลาที่ใช้ในการทำงานภายในตัวลูปทุกกระบวนความ

2.4 HTML

สิ่งที่น่าสนใจมากที่สุดในการเล่นอินเตอร์เน็ตก็คือ การใช้บริการ World Wide Web หรือ เรียกว่า WWWความนิยมของบริการนี้ได้ขยายตัวไปอย่างรวดเร็ว ซึ่งผู้ใช้บริการไม่ต้องพึ่งผู้เชี่ยวชาญก็สามารถใช้บริการได้
เพราะมีง่ายและ สะดวกในการใช้งาน ส่วนที่สำคัญอีกสองส่วนในการใช้บริการ WWW ก็คือตัวโปรแกรมที่เรียกว่า browser หรือ Web browser และแหล่งข้อมูลหรือ Web site ที่จะคอยให้บริการผู้ใช้ Web site หรือ Web system หรือ Web server ก็คือระบบคอมพิวเตอร์ที่ให้บริการเรียกค้นข้อมูลแบบหนึ่งในอินเตอร์เน็ต แต่การที่จะเรียกค้นข้อมูลได้นั้นต้องอาศัยโปรแกรมพิเศษที่เรียกว่า Web browser ในการเรียกดู ข้อมูลจะอยู่ในรูป ข้อความ เสียง และมีทั้งภาพเคลื่อนไหว ซึ่งเราเรียกเอกสารดังกล่าวว่า Home page หรือ Web page ส่วนสถานที่ๆข้อมูลนั้นอยู่เราเรียกว่า Web site

ในการจัดตั้ง Web site ซึ่งเป็นเครื่องคอมพิวเตอร์เครื่องหนึ่งที่เชื่อมต่อเข้าในอินเตอร์เน็ต ก็จะต้องมีหมายเลข IP address สำหรับอ้างอิง และผู้ที่จัดตั้ง Web site ก็จะต้องขอลงทะเบียนชื่อ Web site ของตนอีกครั้ง ชื่อที่ว่านี้คือ Domain name ที่ใช้เรียกอ้างอิงแทนหมายเลข IP address การจัดตั้งชื่อ Domain name นั้นมีความสำคัญเนื่องจากจะเป็นส่วนหนึ่งของข้อมูล URL (Uniform Resource Locator) เพราะโปรแกรม Web browser จะใช้ข้อมูล URL นี้ในการค้นหาที่อยู่ของ Web site การติดต่อกันในอินเตอร์เน็ตใช้โปรโตคอล TCP/IP ซึ่งมีการทำงานที่เรียกว่า Domain name system จะคอยแปลงชื่อ Domain ให้เป็นหมายเลข IP อีกทีหนึ่งแล้วจึงนำค่า IP address ไปอ้างอิงเพื่อส่งถ่ายข้อมูลต่อไป

ตัวอย่างเช่น http://www.ibm.com/Welcome/screen1.htm

* ตัวอักษร http ย่อมาจาก HyperText Protocol เป็นข้อมูลที่บอกให้ทราบว่าเป็นการใช้บริการชนิดใดของอินเตอร์เน็ต สำหรับ http เป็นบริการดูข้อมูล WWW ซึ่งมีข้อมูลเป็น HyperText นั่นเองถ้าเป็น ftp จะหมายถึงการใช้บริการการโอนถ่ายไฟล์ข้อมูล(File Transfer Protocol)

* ตัวอักษร www.ibm.com เป็นชื่อ Domain name หรือเรียกว่าชื่อตำแหน่งที่จะไปดูข้อมูลซึ่งถูกแปลงเป็นตัวเลข IP address โดยกลไก Domain name system นั่นเอง

* ตัวอักษร Welcome เป็นชื่อไดเรกทอรี่ที่เก็บ home page หน้าจอแรกของ Web site นี้ซึ่งอาจเป็นชื่ออะไรก็ได้แล้วแต่จะตั้ง

* ตัวอักษร screen1.htm เป็นข้อมูลหน้าจอแรกที่จะแสดงเมื่อผู้ใช้บริการเรียกเข้ามาดู

2.4.1 WEB PAGE และ HTML

ในการจัดเตรียมข้อมูลที่อยู่ในรูปแบบ Web page นี้ทำได้ 2 วิธีคือ วิธีแรกจะเริ่มต้นโดยการเขียนโปรแกรมในภาษาที่มีชื่อว่า Hypertext Markup Language (HTML) ซึ่งเป็นภาษาที่ใช้ในการจัดทำหน้าตาของข้อมูลที่จะนำเสนอที่นอกจากจะมีลักษณะของ hypertext แล้วยังสามารถแสดงรูปกราฟิก เสียง และภาพ วีดีโอได้อีกด้วย และสามารถจัดให้มีลักษณะการนำเสนอข้อมูลเป็นลำดับชั้น เชื่อมโยงไปยังที่อื่นๆ อีกก็ได้ส่วนวิธีที่สอง คือการใช้โปรแกรมช่วยสร้าง Web page ในการแปลงเอกสารรูปแบบต่างๆให้กลายเป็นภาษา HTML ขึ้นมาโดยไม่ต้องเขียนโปรแกรมขึ้นมาเอง โปรแกรมประเภทนี้เรียกว่าเป็น HTML Editor หรือ HTML Authoring Tool หรือแม้แต่เป็นโปรแกรมทั่วๆไป เช่น Microsoft Office รุ่นใหม่ก็สามารถสร้างเอกสารที่เป็น HTML ได้ทันทีในแอพพลิเคชั่นแต่ละตัวเลย และ Word,Exel,Powerpoint

HTML เป็นภาษาสั่งงานแบบหนึ่งใช้สำหรับจัดการแสดงผลข้อมูลที่ถูกจัดเก็บใน Web site หรือ Web server ตามปกติแล้วข้อมูลที่จะนำมาแสดงเป็นข้อมูลแบบ HyperText ภาษาสั่งงาน HTML จะมีส่วนที่เรียกว่า Markup tag (ส่วนขยายลักษณะหรือเรียกย่อๆ ว่า tag) ที่จะคอยบอกให้โปรแกรม Web browser ได้ทราบว่า หน้าจอหรือข้อความนั้นๆ จะถูกแสดงอยู่ในลักษณะอย่างไร ส่วนขยายลักษณะหรือ Markup tag นี้จะเป็นข้อความหรือคำสั่งที่มีเครื่องหมาย < > ครอบเอาไว้ และใช้เครื่องหมาย " / "เป็นตัวจบ เช่นขึ้นต้นด้วย <H1>และจบด้วย</H1> ภายใน Web page ที่ถูกสร้างจากภาษา HTML นั้นจะมีส่วนของ Markup tag ในแบบต่างๆกัน เพื่อบอกให้โปรแกรม browser ได้ทราบและแสดงผลได้อย่างถูกต้อง ตัวอย่างของ tag และความหมายของ tag

<HTML> เป็น tag ที่จะปรากฏขึ้นต้นในแต่ละ page เสมอ เพื่อบอกให้ทราบว่าข้อมูลcontent ภายในนี้ถูกเขียนขึ้นในรูปแบบของ HTML และจบที่ tag </HTML> แสดงว่าจบข้อมูลหรือ จบเอกสารนี้แล้ว

<HEAD> เป็น tag ที่บอกว่าข้อความต่อไปนี้เป็นหัวเรื่องและจบด้วย tag </HEAD>

<TITLE> เป็น tag ที่จะบอกว่าข้อความต่อไปนี้เป็นชื่อไตเติ้ลของ page นี้ ซึ่งจะแสดงอยู่ที่ส่วนแสดงไตเติ้ลของ โปรแกรม Web browser เมื่อใช้งานที่หน้าจอนี้ และจบด้วย</TITLE>

<BODY> เป็น tag ที่บอกว่าส่วนของข้อความทั้งหมดรวมทั้งรูปภาพ จะเป็นส่วนของเนื้อหาข้อมูลใน page นี้

<P> เป็น tag ที่จะแบ่งแต่ละย่อหน้า (paragraph) ออกจากกัน

 เป็น tag ที่บอกให้ขึ้นบรรทัดใหม่

<B> และ <I> เป็น tag ที่บอกให้แสดงอักษรเป็นตัวหนา หรือตัวเอียง ตามลำดับ

ภาษา HTML ได้พัฒนามาตลอดจนปัจจุบันนี้เป็นเวอร์ชั่น3.2 แล้ว สิ่งที่เพิ่มเข้ามาก็คือ tag ซึ่งมี tag <CENTER> และ <BLINK> ซึ่งโปรแกรม browser netscape navigator สามารถแสดงผล tag นี้ได้ แต่โปรแกรม browser ซึ่งบางตัวจะไม่รู้จัก tag 2 ชนิดนี้

2.4.2 URI และ URL

สำหรับ URI ย่อมาจาก Uniform Resource Identifiers ส่วน URL ย่อมาจาก Uniform Resource Locator คือข้อกำหนดการอ้าง ตำแหน่งของข้อมูลที่จะมาแสดงบนบราวเซอร์ หรืออ้างถึงวิธีในการแสดงข้อมูลบนบราวเซอร์ผ่านทางโปรแกรมต่างๆ

ข้อแตกต่างระหว่าง URI กับ URL คือ URI เป็นข้อกำหนดใหม่ที่มีเนื้อหาสาระครอบคลุมข้อกำหนด URL อีกทีหนึ่ง โดยมีส่วนเพิ่มเติมที่สำคัญ คือ ชื่อไฟล์ ทำให้ผู้ใช้สามารถอ่านไฟล์ที่เป็นภาษาอื่นๆที่ไม่ใช่ภาษาอังกฤษได้ ซึ่งรูปแบบในการเขียน URI หรือ URL จะประกอบไปด้วย 3 ส่วนคือ

protocol://server_name/file_name

1. protocol ใช้ระบุวิธีการที่เซิร์ฟเวอร์จะส่งข้อมูลให้กับบราวเซอร์ซึ่งปกติจะเป็น http
2. server_name คือชื่อของเครื่องที่จะติดต่อขอข้อมูล
3. file_name คือ ตำแหน่งและชื่อไฟล์ที่ต้องการเรียกใช้ โดยต้องคั่นแต่ละไดเรกทอรี่ด้วยเครื่องหมาย /

2.4.3 URL แบบเต็มและแบบย่อ

การอ้างถึงไฟล์ HTML แบบเต็ม ก็ยกตัวอย่างได้เช่น http://www.microsoft.com/dhtml เป็นต้น URL แบบนี้จะระบุส่วนประกอบเต็ม 3 ส่วนวิธีนี้อาจไม่สะดวกนักหากไฟล์ HTML ใหม่ที่จะอ้างถึงเป็นไฟล์ที่อยู่ในเครื่องเดียวกันกับไฟล์เดิม ผู้พัฒนาเว็บจะต้องเขียนคำว่า www.microsoft.com ซ้ำกันทุกๆครั้ง ดังนั้น URL จึงยอมให้อ้างอิงถึงไฟล์ HTML แบบย่อ(relative)ได้

บทที่ 3
วิธีดำเนินการวิจัย

3.1 ขั้นตอนในการดำเนินการวิจัย

ในบทนี้จะกล่าวถึงการรับ URL ที่เข้ามาเก็บเป็นโครงสร้างในรูปแบบของ Sequential และ Binary เพื่อทำการเปรียบเทียบ โดยมีข้อกำหนด ดังที่จะกล่าวถึงดังต่อไปนี้

เมื่อมี input เข้ามาซึ่ง input ก็คือ URL ที่ต้องการเรียกใช้ และ ข้อมูลที่เข้ามาเก็บในแต่ละโหนดจะประกอบไปด้วย

- ลักษณะการทำงานของโปรแกรมตามที่ได้ออกแบบไว้
  - ชื่อ URL ที่ต้องการทำการเรียกใช้เวลา เพื่อที่จะสามารถดูได้ว่า URL ไหนที่ ถูกเรียกใช้เมื่อเวลาเท่าใด และสามารถที่จะลบ URL นั้น
  - เมื่อ cache เต็ม จะลบ URL ที่ไม่ถูกเรียกใช้มานานทิ้งไป และเก็บ URL ใหม่ลงไปใน cache
  - วันที่ ของการเรียกใช้ URLเมื่อ cache เต็ม จะใช้หลักการ least recent use ในการเลือก URL ที่จะลบทิ้ง นั่นคือ จะลบ URL ที่ ถูกเรียกใช้น้อยที่สุด

- ข้อกำหนด
  1. โดยเฉลี่ย page ที่เก็บใน cache มีขนาด 1-3 kb/page (ซึ่งเป็นขนาดจริง)
  2. จะมีวิธีในการคิดจำนวน page ที่จะเก็บ URL ใน cache คือ สมมุติให้แต่ละ page ที่เก็บมีขนาด 2 kb/pageซึ่งเป็นขนาดโดยเฉลี่ย และ สมมุติว่าเรากำหนดขนาด cache คือ 60 kb คือ นำขนาดของ cache มาหารด้วย 2 จะได้จำนวน page

     ฉะนั้น cache จะเก็บได้ 60/2 = 30 page

ในการที่จะเปรียบเทียบประสิทธิภาพของ cache ที่ได้สร้างขึ้นนั้น ผู้จัดทำ จึงได้นำวิธีการเก็บข้อมูลแบบ sequential มาเปรียบเทียบกับ การเก็บแบบ binary tree ซึ่งจะทำการคิด Big(o) เพื่อเปรียบเทียบ

3.1.1 การออกแบบโครงสร้างข้อมูลของ Binary Tree

จากการศึกษาโครงสร้างข้อมูลแบบต่างๆ แล้วนั้น เราจึงได้รู้ว่าควรใช้ Binary Tree ในการจัดเก็บ URL เพื่อแก้ปัญหาที่เราสนใจ ทั้งนี้เพราะว่า Binary Tree นั้น เหมาะกับการที่จะใช้จัดเก็บและค้นหาข้อมูลซึ่งใช้พื้นฐานในการจัดเก็บและการค้นหาที่ง่ายแต่มีประสิทธิภาพ

โดยเฉพาะอย่างยิ่ง เมื่อเปรียบเทียบกันโดยใช้พื้นฐานทางด้านความซับซ้อนของอัลกอริทึม ซึ่งการใช้โครงสร้างข้อมูลแบบ Binary Tree นั้นมีประสิทธิภาพดีที่สุด

ความซับซ้อนของอัลกอริทึมของ Binary Tree $O(n) = \log_2 n$

- **โครงสร้างข้อมูลที่ใช้หลักการของ Binary Tree**

โดยรวมแล้วเราจะแบ่งการออกแบบเป็น 4 ส่วนด้วยกันคือ

1 หลักในการสร้าง Tree

2 หลักในการเพิ่มโหนดเข้าไปใน Tree

3 หลักในการลบ Tree

4 หลักในการท่อง Tree

ซึ่งทั้ง 4 ส่วนนี้จะเป็นหลักในการที่จะทำให้โครงสร้างข้อมูลที่เราออกแบบขึ้นมาโดยใช้หลักการของ Binary Tree นั้นมีความเป็นระบบและถูกต้องในการจัดการข้อมูล (URL) ที่จะจัดเก็บ

3.1.1.1 หลักในการสร้าง Tree

จากที่ได้กล่าวมาแล้วว่าในการแก้ปัญหาพิเศษนี้เราจะใช้หลักการของ Binary Tree ในการสร้างโครงสร้างข้อมูลสำหรับจัดเก็บแล้วค้นหา URL โดยจะใช้หลักการของ Binary Tree ที่ว่า **ต้นไม้แบบ Bianry Tree เป็นต้นไม้ที่แต่ละโหนดซึ่งมีต้นไม้ย่อยได้ไม่เกิน 2** นั่นเอง ซึ่งจะมีการในนิยามของต้นไม้ Binary ดังนี้

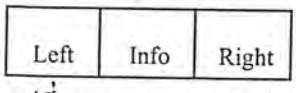
ให้เป็น Binary Tree (T) ซึ่งเป็นเซทของอิลิเมนท์ของ URL ที่เรียกว่าโหนด ซึ่งมี 2 ลักษณะ

1. T ว่างไม่มีข้อมูลเรียกว่า Empty tree
2. T ประกอบไปด้วย โหนด R เรียกว่า รากของ T (root of T) และโหนดที่เหลือของ T จะสร้างคู่ทรีที่แยกย่อยออกไปเป็น Binary Tree T1 และ T2

ถ้า T ประกอบด้วย ราก R จะมีทรี T1 และ T2 ซึ่งเรียกว่า ทรีย่อยซ้าย และทรีย่อยขวา (left and right subtree) ของ R หาก T1 มีข้อมูลบรรจุอยู่จะเรียกว่าเป็น ซัคเซสเซอร์ซ้าย (left successor) ของ R ซึ่งในแต่ละโหนดที่จะใช้เก็บข้อมูล (www) จะประกอบไปด้วย 3 ส่วน นั่นก็คือ

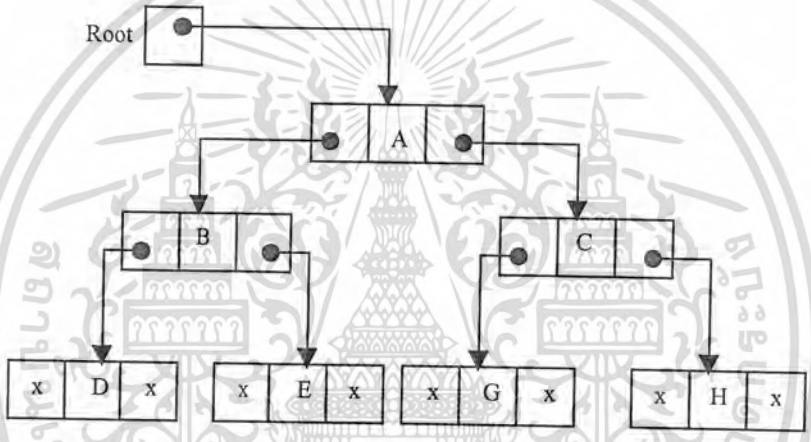
1. Info เป็น Array ซึ่งจะเก็บค่าข้อมูล URL ของโหนด N

2. Left เป็น Pointer ที่ชี้ไปยังตำแหน่งของโหนดลูกด้านซ้ายของโหนด N
3. Right เป็น Pointer ที่ชี้ไปยังตำแหน่งของโหนดลูกด้านขวาของโหนด N



รูปที่ 3.1 แสดงรายละเอียดของแต่ละโหนด

ตัวแปร Root เก็บตำแหน่งของราก R ของ T ถ้าทรีย่อยว่าง ตัวชี้ตำแหน่งจะเก็บค่า null เพราะฉะนั้นถ้าทรี T ว่าง ค่า Root จะเป็น null



รูปที่ 3.2 แสดงการลิงค์ข้อมูลแบบ Binary Tree

ในการจัดการเกี่ยวกับข้อมูลจริงๆ ที่ได้ออกแบบไว้เราจะทำการเก็บ ข้อมูล โดยจะกำหนดให้แต่ละโนดเก็บประโยคส่วนหนึ่งของชื่อ URL โดยจะมีกฎเกณฑ์ในการตัดชื่อของ URL

- **หลักในการแบ่ง URL เพื่อเก็บข้อมูล**

  1. แบ่งตามมหัพภาค(.) เช่น

  - www.chula.ac.th จะแบ่งได้เป็น www , chula , ac , th
  - www.kmitl.ac.th จะแบ่งได้เป็น www , kmitl , ac , th
  - www.yahoo.com จะแบ่งได้เป็น www , yahco , com

ซึ่งในโหนดของข้อมูล Info จะเก็บข้อความของ URL แต่ละคำที่ตัดได้ตามกฎที่ได้กล่าวได้แล้ว โดยจะมีกฎการเก็บว่าคำๆ ไหนใน URL ควรจะอยู่ในโนดที่เท่าไร โดยเราจะกำหนดโดย

- **หลักการเก็บคำของ URL**

1. เก็บจากคำหลังมายังคำหน้า เช่น

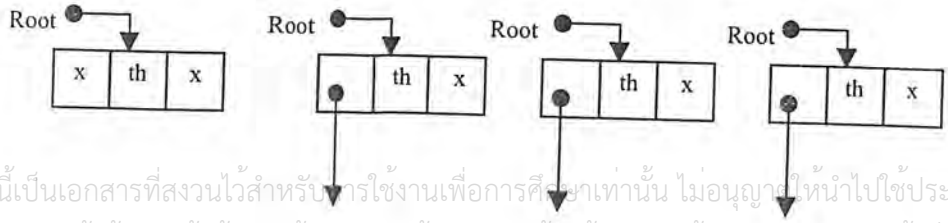
- www.chula.ac.th จะแบ่งได้ เป็น www , chula , ac , th เวลาเก็บก็จะเป็น th → ac → chula → www
- www.kmitl.ac.th จะแบ่งได้เป็น www , kmitl , ac , th เวลาเก็บก็จะเป็น th → ac → kmitl → www
- www.yahoo.com จะแบ่งได้เป็น www , yahoo , com เวลาเก็บก็จะเป็น com → yahoo → www

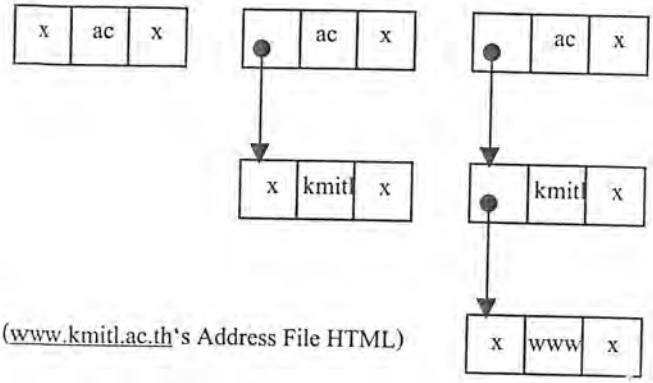
2. ถ้าเป็นตัวแรกของ Tree จะเป็น Root เช่น

- www.chula.ac.th จะแบ่งได้ เป็น www , chula , ac , th เวลาเก็บก็จะเป็น th → ac → chula → www เพราะฉะนั้น th จะเป็น Root
- www.yahoo.com จะแบ่งได้เป็น www , yahoo , com เวลาเก็บก็จะเป็น com → yahoo → www เพราะฉะนั้น com จะเป็น Root

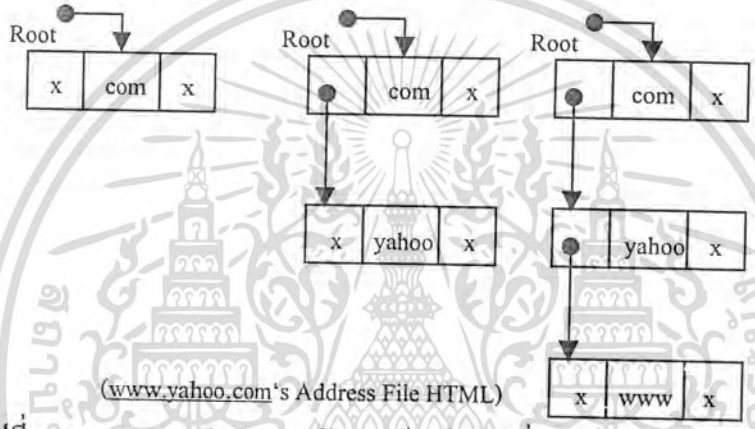
3. โหนดสุดท้าย (Leave Node) จะเก็บที่อยู่ข้อมูลของ URL ตาม path Tree นั้น เช่น

- www.chula.ac.th จะแบ่งได้ เป็น www , chula , ac , th เวลาเก็บก็จะเป็น th → ac → chula → www โหนดสุดท้ายที่ได้จากการท่องของนี้ก็จะเก็บที่อยู่ของไฟล์ www.chula.ac.th ในฮาร์ดดิสก์
- www.yahoo.com จะแบ่งได้เป็น www , yahoo , com เวลาเก็บก็จะเป็น com → yahoo → www โนดสุดท้ายที่ได้จากการท่องของนี้ก็จะเก็บที่อยู่ของไฟล์ www.yahoo.com ในฮาร์ดดิสก์





รูปที่ 3.3 แสดงการทำงานของโครงสร้างข้อมูล เมื่อเรียกใช้ www.kmitl.ac.th



รูปที่ 3.4 แสดงการทำงานของโครงสร้างข้อมูล เมื่อเรียกใช้ www.yahoo.com

อีกอย่างซึ่งมีความสำคัญในการทำงานของโครงสร้างที่ออกแบบมานี้ก็คือกฎของการลิงค์ ซึ่งจะเป็นส่วนที่จะกำหนดการท่อง Tree นี้เพื่อค้นหาอีกด้วย ซึ่งมีกฎดังนี้

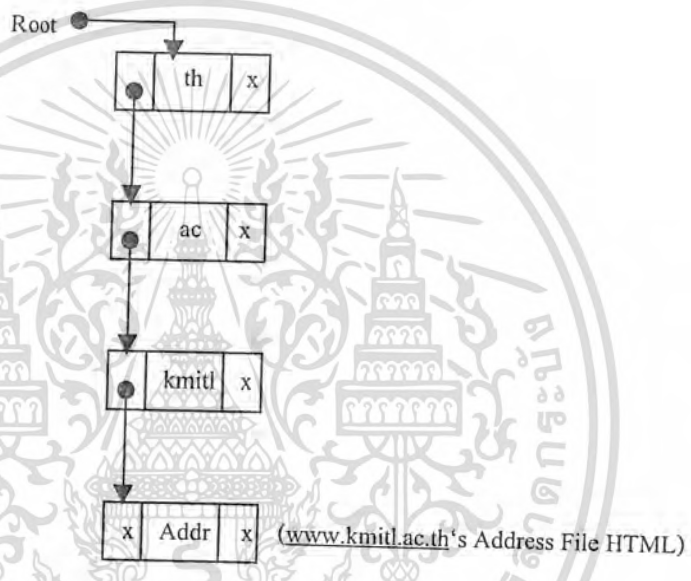
3.1.1.2 หลักในการเพิ่มโหนดเข้าไปใน Tree

- จะใช้การเพิ่มโหนดเข้าไปเมื่อใด

เราจะทำการเพิ่มข้อมูลในโครงสร้างข้อมูลของเราก็ต่อเมื่อ มีการเรียกใช้ URL เกิดซึ่งในซึ่ง URL ที่เรียกเพิ่มเข้ามานี้จะต้องเป็น URL ไม่ถูกเก็บใน Cache ของระบบโดย เมื่อมีการเรียก URL เข้ามาก็จะทำการแบ่งคำของ URL ตามที่ได้กล่าวไว้แล้ว และจะทำเช็คดูว่าในแต่ละคำนั้นมีอยู่ใน Tree ของ Cache หรือยัง ถ้ามีก็จะทำการเปรียบเทียบคำต่อไป ทำอย่างนี้ไปเรื่อยๆ จนหมดคำของ URL ซึ่งเป็นการแสดงว่า URL นั้นมีอยู่ใน Tree ของ Cache ของระบบแต่ถ้าเกิดเมื่อเปรียบเทียบกันแล้วไม่มีอยู่ใน Tree ก็จะทำการเพิ่มคำๆ นั้นของ URL ใหม่ลงไปใน Tree โดยวิธีการจะได้

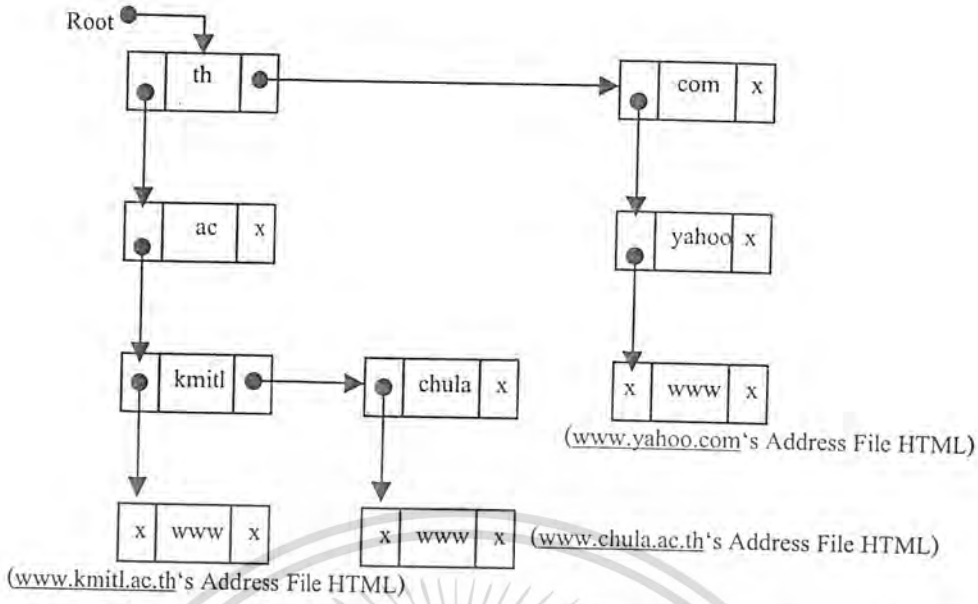
กล่าวต่อไป ซึ่งการเพิ่ม URL เข้ามาใน Tree จะต้องมีการจัดการที่เป็นระบบ ซึ่งทำให้ค้นหาข้อมูล HTML ของแต่ละ URL ได้ไม่ผิดพลาด ซึ่งจะสอดคล้องกับการค้นหาอีกด้วย

- ในการใช้ **Pointer ลิงค์ทางด้านซ้าย(Left[k])**
  - เราจะใช้ลิงค์ทางด้านซ้ายต่อไปยังโหนดต่อไปก็ต่อเมื่อ ตัวของข้อมูลยังไม่หมดและลิงค์ทางด้านซ้ายยังไม่เก็บตำแหน่งข้อมูล(**ไม่มีข้อมูลต่อไปแล้ว**)เช่น www.kmitl.ac.th



รูปที่ 3.5 แสดงการใช้ลิงค์ทางด้านซ้าย

- ในการใช้ **Pointer ลิงค์ทางด้านขวา (Rigth[k])**
  - เราจะใช้ลิงค์ทางด้านขวาก็ต่อเมื่อในการเปรียบเทียบคำของ URL ใหม่ที่ได้กับตัวโหนดเดิมว่ามีหรือยัง ถ้ามีแล้วก็จะทำการลิงค์ทางด้านซ้ายซึ่งมีอยู่แล้ว ถ้ายังไม่มีก็จะทำการลิงค์ทางด้านขวาถ้าทางด้านขวาก็มีข้อมูลอยู่(คำของ URL)ก็จะทำการลิงค์ไปทางด้านขวาของโหนดที่ไม่ได้ลิงค์ทางด้านขวาไว้ เช่นถ้าเพิ่ม www.chula.ac.th กับ www.yahoo.com ก็จะได้ Tree ที่อยู่ในที่ 3.6



รูปที่ 3.6 แสดงรูปการทำงาน Binary Tree ในการใช้ลิงค์ทางด้านขวา เมื่อมีการเรียกใช้URLwww.chula.ac.th กับ www.yahoo.com เข้าไป

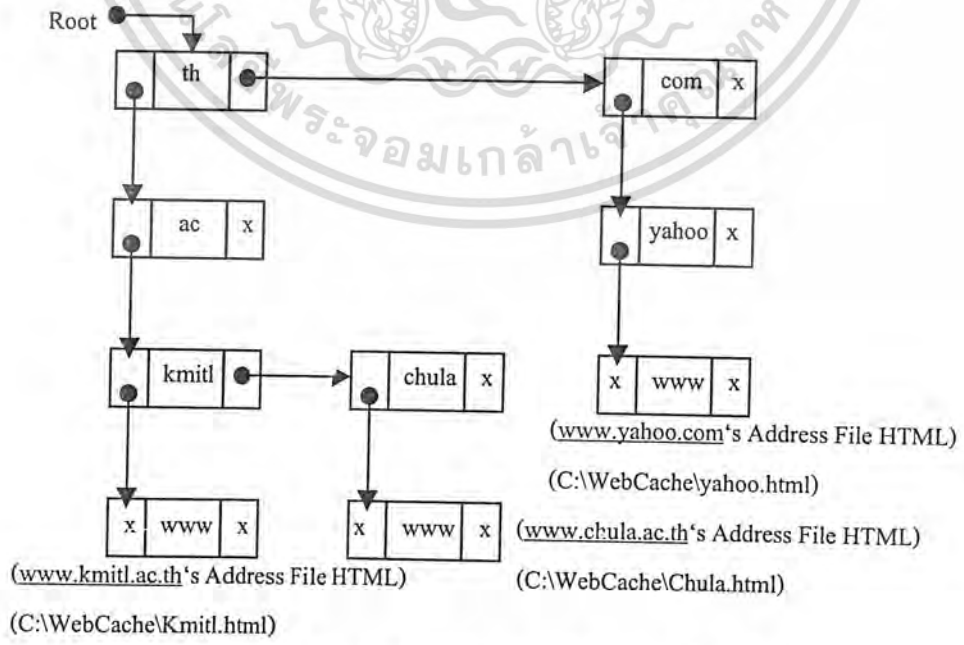
เมื่อมีการเพิ่มข้อมูลเข้าไปใน Tree แล้ว ในความจริงทางการใช้งานแล้ว จะหมายความว่าได้มีการเรียกใช้ URL นี้แล้วซึ่งเราจะใช้กฎที่เรากำหนดขึ้นใน Tree ของเรานี้ เพื่อเป็นการกำหนดว่าควรจะมี URL และ ไฟล์ HTML อยู่ในระบบได้เท่าใด ทั้งนี้เพราะ Cache ของเรานั้นได้กำหนดขนาดซึ่งจะมีขนาดที่จำกัดตามขนาดที่เรากำหนด

3.1.1.3 หลักในการลบข้อมูลออกจาก Tree

จากที่ได้กล่าวไว้ในหัวข้อก่อนหน้านี้แล้วว่าเมื่อข้อมูลใน Tree เริ่มจะเต็มความจุของ Cache แล้วก็จะเกิดการลบข้อมูลออกจาก Tree ซึ่งก็จะมีหลักการทำงานที่แน่นอนเพื่อความเป็นระเบียบในการจัดการข้อมูล ซึ่งลักษณะต้นไม้ที่เราใช้จะเป็นต้นไม้ Binary แบบไดนามิกซึ่งจะใช้ Pointer ในการลิงค์ข้อมูลต่างๆ ฉะนั้นในการลบข้อมูลที่เกิดขึ้นเราจึงใช้หลักการลบโหนดใน Binary เข้ามาช่วยในการลบด้วย โดยเราจะดูว่าชื่อ URL ที่ต้องการลบคือ URL ไหน โดยดูจากวิธีการในหัวข้อข้างต้น แล้วจะทำการดูว่า URL นั้น อยู่สาย Tree แล้วจะทำการลบโหนดของ URL นั้น ๆ ที่ไม่เกี่ยวข้องกับ URL อื่นๆ ถ้าเกิดว่ามีการใช้ชื่อของ URL ร่วมกันจะไม่มีการลบชื่อ URL ที่มีการใช้ร่วมกันนั้นออกไป **จะลบเพียง URL ที่ต้องการลบและไม่เกี่ยวข้องกับ URL** อื่นเท่านั้น โดยเราสามารถดูได้ว่าโหนดแต่ละโหนดจะมีความเกี่ยวข้องกับกับข้อมูลที่

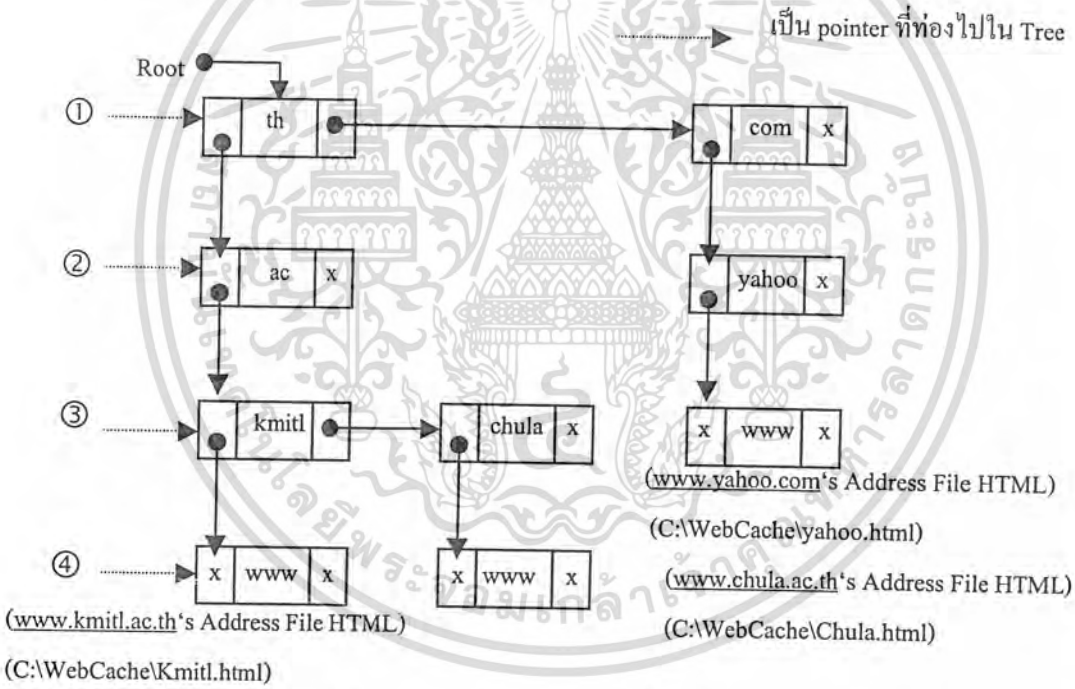
ต้องการลบหรือไม่ ก็โดยดูจากลิงค์ของ Pointer ในแต่ละข้างของแต่ละโหนดว่ามีค่าเป็นอย่างไร ในทางปฏิบัติแล้วเราจะเริ่มการลบข้อมูล โดยจะเริ่มจาก Leaf Node ซึ่งจะเก็บค่าของ Hit Rate กับ Address ของข้อมูล และจะทำการดูที่ Parent Node ว่า Pointer ทั้ง 2 ข้างนั้นใช้ลิงค์ไปที่อื่นหรือไม่ หรือไม่มีค่าเป็น Null ซึ่งจะหมายความว่า โหนดนั้นได้มีการใช้บางส่วนร่วมกับโหนดอื่นๆ ซึ่งจะลบไม่ได้ จะลบได้ก็ต่อเมื่อ Parent Node ที่ต้องการจะลบนั้นมีการใช้ลิงค์แค่ลิงค์ใดลิงค์หนึ่งเท่านั้น ซึ่งจะไม่มีส่วนเกี่ยวข้องกับโหนดอื่นๆ ซึ่งในการลบโหนดจริงๆ นั้นจะเป็นดังนี้

โดยจะไปที่ Parent Node ของโหนดนั้นๆ แล้วจะทำการปล่อย Pointer ให้ลอย ไม่ไปลิงค์ที่โหนดข้อมูลอีกต่อไป แล้วถึงจะทำการลบโหนดนั้นๆ ออกไป ในการลบโหนดในโครงสร้างข้อมูลที่เราออกแบบขึ้นนั้นจะต้องใช้การท่องหรือค้นหาข้อมูลใน Tree ซึ่งในการท่อง Tree นั้นเราจะต้องใช้ Pointer ท่องไปใน Tree โดยเริ่มตั้งแต่ Root แล้วจึงไล่ลงมาเรื่อยๆ จาก Root ลงมาโดยจะลงมาตาม Pointer ที่ลิงค์ตามโหนดต่างๆ ในการลบนั้นจะต้องลบจาก Leaf Node เท่านั้น ทั้งนี้เพราะว่าในการลบอย่างที่ได้กล่าวไว้แล้วว่าจะต้องทำการปล่อย Pointer ที่ชี้ไปยัง โหนดออก แต่ถ้าโหนดที่ถูกปล่อย Pointer ออกไปนั้นไม่เป็น Leaf Node ก็จะทำให้ข้อมูลที่อยู่ต่อจากโหนดนั้นๆ ถูกลบออกไปด้วย ฉะนั้นในการลบโหนดใน Tree จึงควรกำหนดให้ลบตั้งแต่ที่Leaf Node ขึ้นไปเรื่อยๆ จนถึงโหนดที่เราไม่ต้องการจะลบแล้ว ซึ่งก็จะเป็นโหนดที่มีการใช้ข้อมูลหรือชื่อ URL ซ้ำกัน ซึ่งไม่สามารถลบโหนดนั้นได้เพราะเป็นการใช้ร่วมกัน

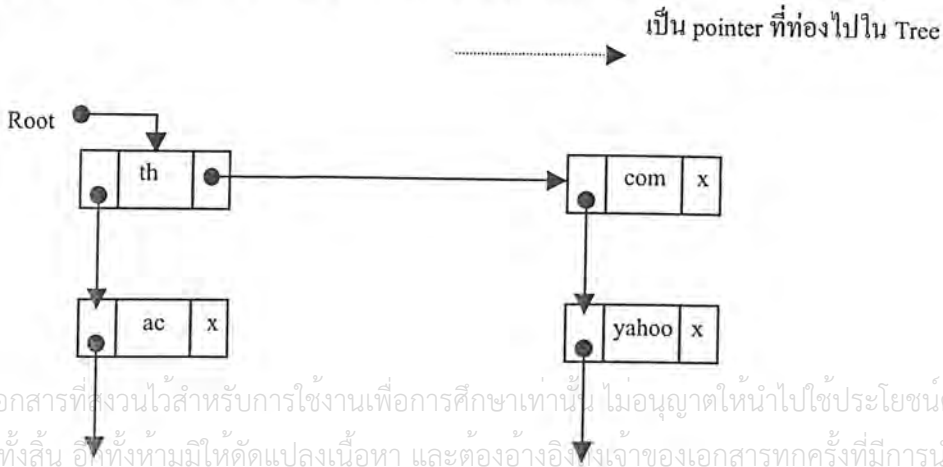


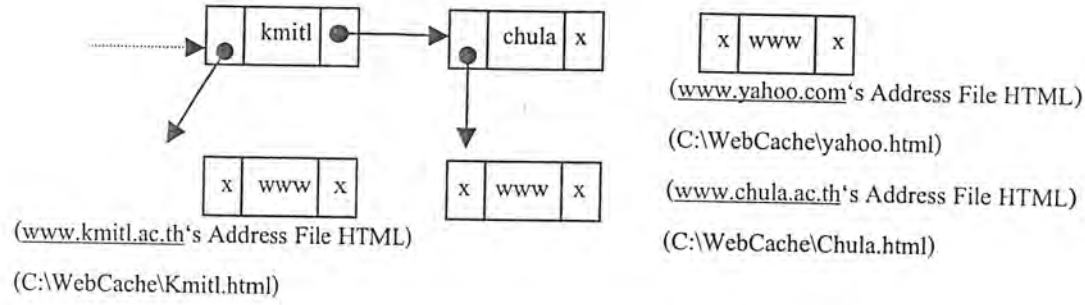
รูปที่ 3.7 แสดงโครงสร้างข้อมูลโดยสมมุติว่า Cache เต็มแล้ว

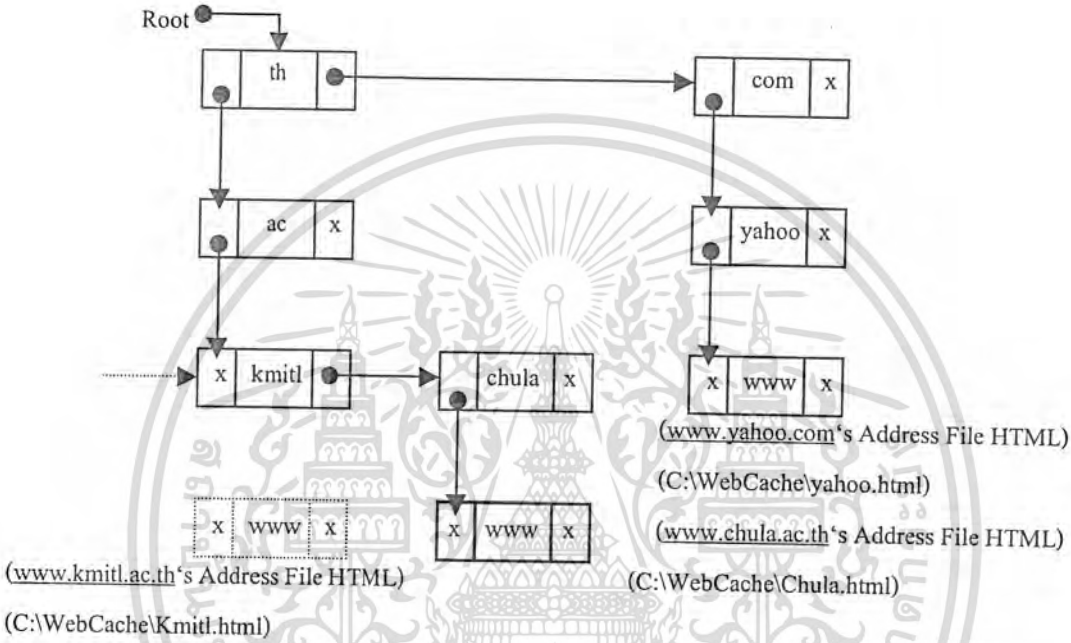
จากรูปตัวอย่างที่ 3.7 จะเป็นการสมมุติว่ามีข้อมูลอยู่ใน Cache คือ www.kmitl.ac.th , www.chula.ac.th และ www.yahoo.com และจะสมมุติว่าในตอนนี้มีการกำหนดให้ Cacheมีขนาดที่จะเก็บข้อมูลได้ 3 URL เท่านั้น ซึ่งจะหมายความว่าตอนนี้ Cache ของระบบได้เต็มแล้วทำให้จะต้องมาดูว่าในตอนนี้ URL ใดมีค่าการถูกเรียกใช้หรือ Hit Rate น้อยที่สุด ก็จะท่องไปใน Tree โดยใช้ Pointer ท่องไปโดยเริ่มจาก Root และเริ่มท่องไปตาม URL ที่มีค่าน้อยที่สุด จนไปถึง Leaf Node หรือโหนดสุดท้าย จากรูปก็จะเห็นว่า แต่ละ URL มีข้อมูลในส่วนที่ Leaf Node ในแต่ละ URL ของตน ซึ่ง URL ที่มีค่า Hit Rate น้อยที่สุดก็คือ www.chula.ac.th ก็จะทำการท่องไปจนถึง Leaf Node จากนั้นก็จะทำการไปที่โหนดพ่อ (Parent Node) แล้วทำการปล่อยโหนดนั้น โดยการปล่อย Pointer ที่ลิงค์ไปยัง Leaf Node ออกไป ดังรูปที่ 3.8



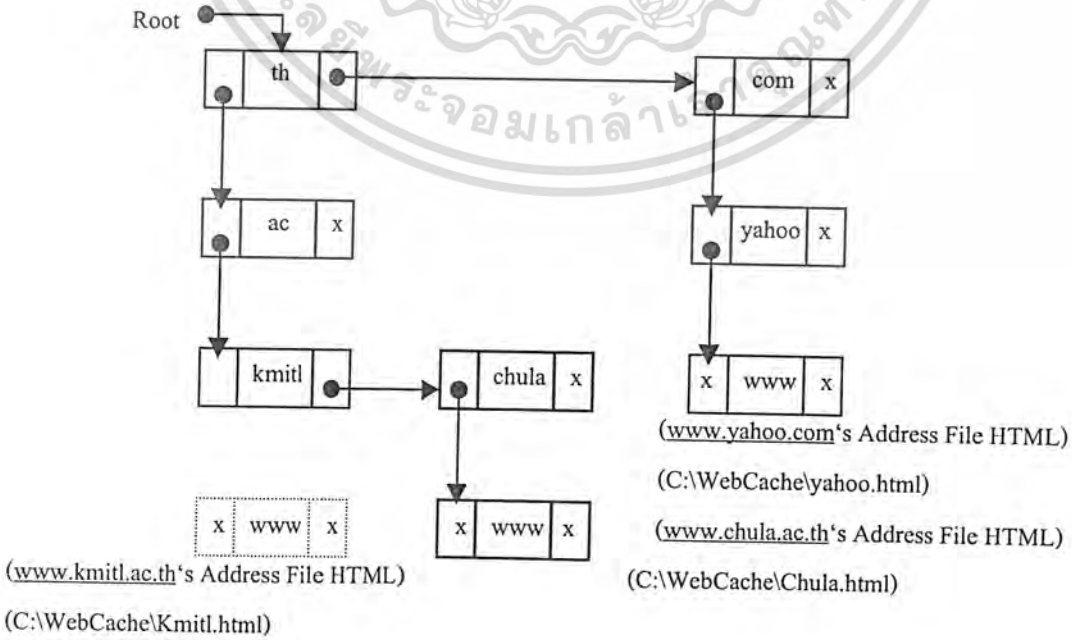
รูปที่ 3.8 แสดงการใช้ Pointer ท่องมาใน Tree เพื่อทำการลบ(www.kmitl.ac.th)



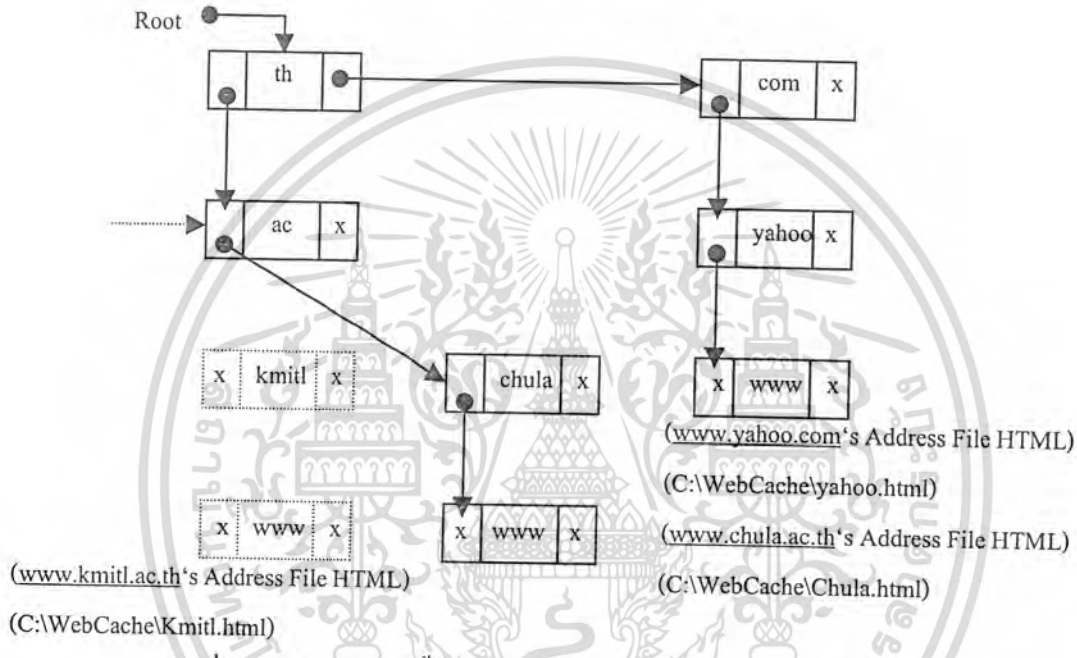




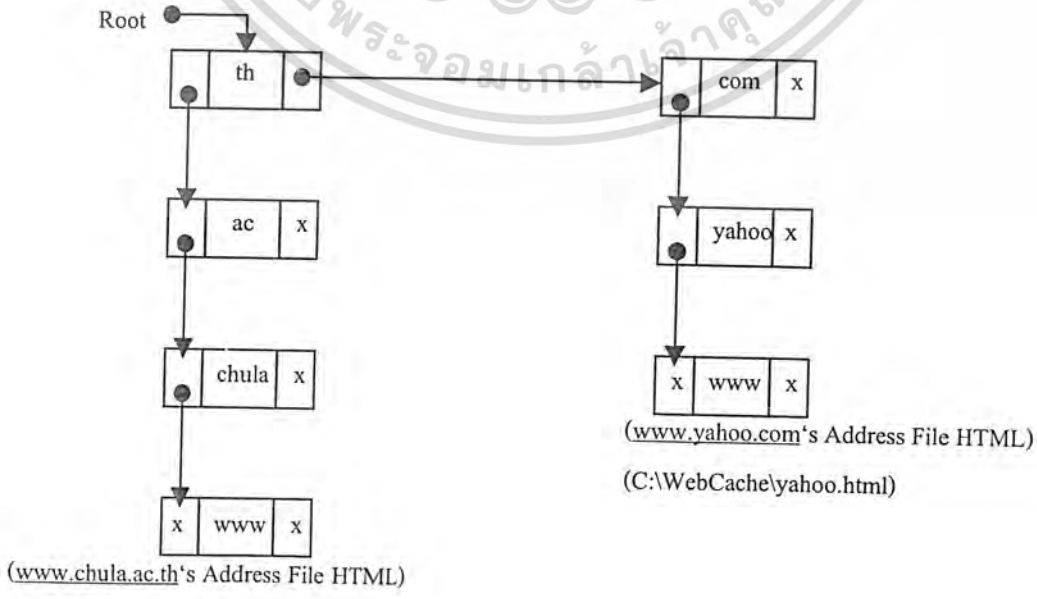
รูปที่ 3.9 แสดงขั้นตอนการลบโหนด URL (www.kmitl.ac.th)



รูปที่ 3.10 แสดงขั้นตอนการลบโหนด URL (www.kmitl.ac.th)



รูปที่ 3.10(ต่อ) แสดงขั้นตอนการลบโหนด URL (www.kmitl.ac.th)



(C:\WebCache\Chula.html)

รูปที่ 3.11 แสดงรูป Tree เมื่อลบโหนด URL (www.kmitl.ac.th) แล้ว

3.1.1.4 หลักในการท่องไปใน Tree ของ Cache

หลักในการท่องไปใน Tree เป็นส่วนหนึ่งที่สำคัญอีกส่วนหนึ่งไม่แพ้ส่วนอื่นๆ ที่ได้กล่าวมา ทั้งนี้ในการที่เราจะสามารถที่จะรู้ได้ว่าข้อมูลใน Tree ของ Cache นั้นมีอะไรบ้าง และการที่ข้อมูลที่ถูกต้องตามที่เราต้องการ ซึ่งตามหลักการของ Binary Tree นั้นจะมีการท่อง Tree อยู่ 3 แบบคือ

1. พรีออเดอร์ (Preorder Traversing)
2. อินออเดอร์ (Inorder Traversing)
3. โพสออเดอร์ (Postorder Traversing)

แต่ตามโครงสร้างของเราที่ได้ออกแบบและรวมถึงการจัดเก็บข้อมูลและการเข้าถึงข้อมูลทำให้เราไม่สามารถใช้การท่อง Tree ทั้ง 3 นี้ได้ เราจึงได้กำหนดการท่องไปใน Tree ขึ้นใหม่ โดยพื้นฐานการท่องไปใน Tree นั้น จะอาศัย Pointer 1 ตัว หรือมากกว่า เพื่อใช้ในการชี้แต่ละโหนดเพื่อท่องไปใน Tree โดยการท่องนี้จะเริ่มจาก Root และในการท่องไปใน Tree ก็จะแบ่งได้เป็น การใช้ Pointer ท่องไปใน Tree ทางซ้ายของ Node และทางขวาของ Node ซึ่งการที่จะใช้การท่องไปในทางไหนนั้นก็ขึ้นกับความสัมพันธ์ของข้อมูล URL โดยในการท่องไปใน Tree เราจะกำหนดให้ท่องลงมาทางด้าน ซ้ายก่อน ถ้าทางด้านซ้ายไม่ใช้ก็จึงจะไปยังทางด้านขวาของโหนด ทั้งนี้เมื่อคิดในกรณีที่น้อยที่สุดในการที่เราจะเจอข้อมูลที่เราต้องการก็จะเป็นการท่องไปตรงๆ ดังนั้นเราจึงกำหนดให้ท่องไปใน Tree ทางซ้ายก่อน โดยในการท่องเพื่อค้นหาข้อมูลจะทำการเปรียบเทียบว่ามี URL ที่เข้ามานี้อยู่ใน Cache หรือไม่ โดยจะทำการตัดชื่อ URL ที่เข้ามาแล้วนำคำแต่ละคำที่เปรียบเทียบ มาเปรียบเทียบในแต่ละโหนด โดยจะเปรียบเทียบคำจากข้างหลังก่อน ถ้ามีใน Cache ก็จะทำการท่องไปทางซ้ายก่อนแล้วจึงเปรียบเทียบตัวถัดไป ถ้าใช้ก็จะทำการท่องไปใน Tree ทางด้านซ้ายต่อไป แต่ถ้าไม่ก็จะไปเปรียบเทียบทางด้านขวา ทำอย่างนี้ไปเรื่อยๆ จนกว่าจะถึง Leaf Node หรือโหนดสุดท้ายโดยดูจากลิงค์ของโหนดทาง 2 ข้างว่าเป็นค่าว่าง หรือ Null หรือไม่ ซึ่งจะแสดงว่าเป็น Leaf Node ก็เป็นอันจบการท่อง Tree ซึ่งจะหมายความว่า URL ที่เข้ามานั้นเป็น URL ที่เคยเรียกใช้มาแล้ว ก็จะทำการเรียกไฟล์ HTML ของ URL นั้นๆ ตาม Address ที่อยู่ที่ Leaf Node และจะทำการบวกค่านับจำนวนการเรียกใช้ไปอีก 1 เพื่อใช้ในการตัดสินใจในการลบ URL เมื่อ Cache เต็ม

แต่ถ้าเกิดท่องไปตามวิธีแล้วไม่เจอตาม URL ที่เข้ามาก็แสดงว่า URL นี้เป็น URL ใหม่ ก็จะทำการดูว่า Cache เต็มหรือยัง ถ้ายังก็จะทำการเพิ่ม URL นี้ไปอยู่ใน Cache โดยวิธีการที่กล่าวมาแล้วข้างต้น

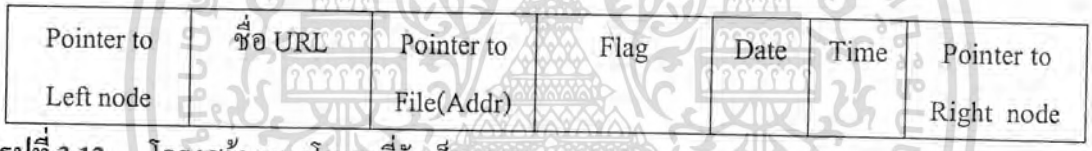
3.1.2 ขั้นตอนการเขียนโปรแกรม

- **การลำดับความสำคัญ**

ในการจัดเก็บURLด้วยรูปแบบBinary Treeนั้นจะใช้ access timeเป็นตัวกำหนดความสำคัญ โดยที่ URLที่มีaccess time นานๆจะมีลำดับความสำคัญต่ำกว่าURLที่มีaccess time น้อยๆ ดังนั้นในการเลือกURLที่จะต้องถูกลบจากcacheจะเป็นURLที่มีaccss timeนานที่สุดซึ่งหมายถึง URL นี้ถูกเรียกใช้มาเป็นเวลานานที่สุด

3.1.2.1 การเก็บ URL แบบ Binary Tree

เป็นการจำลองรูปแบบการจัดเก็บURLโดยใช้โครงสร้างของBinary Tree มาใช้ในการจัดเก็บครงสร้างของ node ที่ใช้เก็บ URL แต่ละ node มีโครงสร้าง ดังนี้



รูปที่ 3.12 โครงสร้างของโหนดที่จัดเก็บURLแบบ binary

- **ในการประกาศโครงสร้าง Tree ใน Delphi**

```
Snode = ^node;
        node = Record
                Name : String[20];
                Addr : String[50];
                TTime : String[5];
                TDate : String[10];
                Flag : Boolean;
                left : Snode;
                right : Snode;
        End;
```

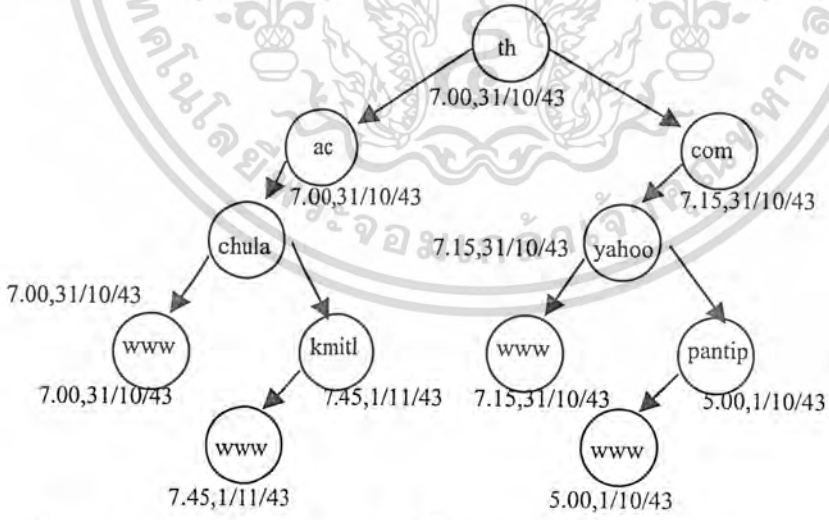
- ข้อกำหนด

1. ณ ตำแหน่ง Root ของแต่ละ URL จะเก็บaccess time ที่นานที่สุดของURLภายใต้ Suffix เดียวกัน
2. ในโหนดๆหนึ่งจะประกอบด้วย
   2.1 pointer to left node จะเป็นการเชื่อมต่อระหว่างโหนด ภายในURLเดียวกัน
   2.2 pointer to right node จะเป็นการเชื่อมต่อระหว่างโหนด ของURLที่ต่างกัน
   2.3 access time จะเป็นเวลาและวันที่ที่ทำการเรียกใช้ URLนั้น

จากตัวอย่างของ Input ที่เข้ามา

URL	TIME	DATE
www.chula.ac.th	7.00	31/10/43
www.yahoo.com	7.15	31/10/43
www.pantip.com	5.00	1/11/43
www.kmitl.ac.th	7.45	1/11/43
www.thairat.co.th	4.00	2/5/44

สามารถแปลงให้อยู่ในรูป Binary tree ได้ดังนี้



รูป 3.13 การแปลงเป็นรูปแบบการจัดเก็บ URL แบบ Binary Tree

สำหรับการเก็บ access time ของ URL ที่ทำการจัดเก็บแบบ Binary จะเก็บ access time ที่น้อยที่สุดของ URL ภายใต้ Suffix เดียวกันเพื่อที่จะได้สะดวกในการค้นหา URL ภายใน Binary นี้ ตัวอย่างเช่น

URL	TIME	DATE
www.chula.ac.th ที่เวลา	7.00	31/10/43
www.kmitl.ac.th ที่เวลา	7.45	1/11/43

จะทำการเก็บ access time ของ www.chula.ac.th คือเวลา 7.00 และวันที่ เดือน และปี 31/10/43 ไว้ที่ node ac และ th และนอกเหนือจากนั้นก็จะทำการจัดเก็บ access time เป็น access time ของ URLที่ได้ทำการเรียกใช้ นั้นเอง

3.1.2.2 Search for URL

การค้นหา URL สำหรับโครงสร้างแบบ Binary จะเกิดขึ้นภายใต้การ insert , delete และ update URL ที่ต้องการ

3.1.2.3 Insert new URL

ในการ insert URL นั้น จะต้องทำการค้นหาข้อมูลดูก่อนว่า ภายใน Binary มี URL ที่ต้องการ insert อยู่หรือไม่ จึงแบ่งได้ 2 กรณี คือ

- insert new URL
- insert URL ที่มีอยู่แล้ว

สำหรับการ insert URL นั้น ก่อนอื่นจะต้องทำการ search ดูก่อนว่าภายใน Binary ที่จัดเก็บนั้นมี URL นี้อยู่หรือไม่ เมื่อ ทำการค้นหาแล้วปรากฏว่า ไม่พบ URL นั้น ก็จะทำการนำ URL ไปต่อท้าย node ที่มี suffix เดียวกัน

แต่ ถ้าเกิดกรณี cache ที่ทำการจัดเก็บ มีขนาดไม่เพียงพอ ดังนั้นจึงต้องทำการ delete URL ที่ไม่จำเป็นทิ้งไป ซึ่งจะกล่าวในหัวข้อ 3.1.2.4

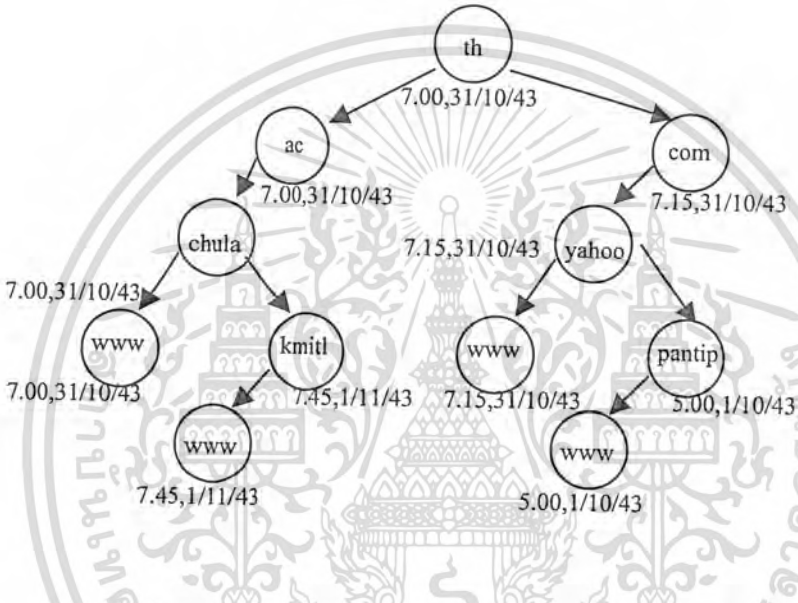
ส่วนการ insert URL ที่มีอยู่แล้วจะทำการ update accesstime ของ node นั้นๆ และทำการจัดเปลี่ยนโครงสร้างของ Binary ให้เรียงลำดับตาม access time ให้ถูกต้องตามข้อกำหนด คือ จากเวลามากไปเวลาน้อยซึ่งจะกล่าวในหัวข้อ 3.1.2.5

3.1.2.4 Delete saved URL

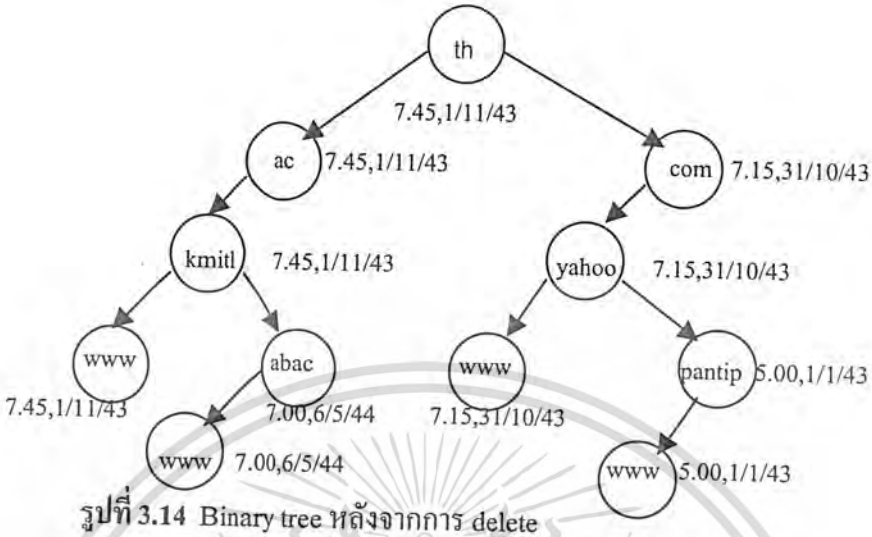
เมื่อต้องทำการ insert new URL แล้ว ปรากฏว่า cache มีเนื้อที่ไม่เพียงพอ ดังนั้น จึงต้องทำการ delete URL ที่มีความสำคัญน้อยที่สุด ซึ่งในที่นี้กำหนดให้ URL ที่มี access time เป็นเวลานาน จะเป็น URL ที่ถูก delete

- **การ Insert ที่มี URL อยู่แล้ว**

ต้องการ Insert URL www.abac.ac.th ที่เวลา 7.00 วันที่ 6/5/44 แล้วปรากฏว่า Cache เต็ม จะต้องทำการ Delete URL ทิ้งจึงต้องทำการ Delete URL ที่มี access time ที่มีค่าน้อยที่สุดทิ้งจากรูป 3.13 (รูปการจัดเก็บแบบ Binary tree)

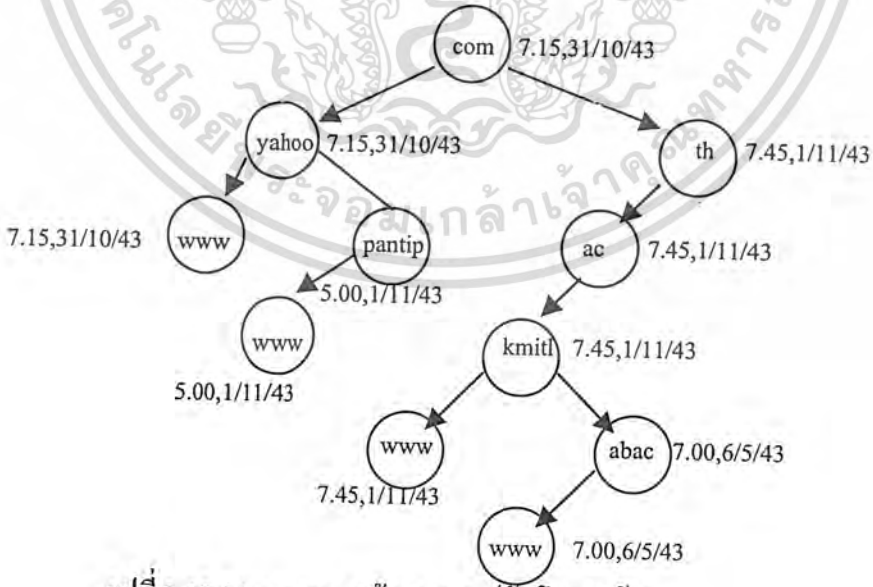


เมื่อพิจารณาจาก access time แล้ว URL ที่จะถูก delete คือ www.chula.ac.th ซึ่งเมื่อทำการ delete www.chula.ac.th ทิ้งไปจะต้องมีการปรับเปลี่ยนโครงสร้างของ Binary tree ใหม่เพื่อที่จะได้ตรงตามข้อกำหนดจะได้ออกมาดังรูป



รูปที่ 3.14 Binary tree หลังจากการ delete

และเมื่อทำการลบ node chula ออกไปจะทำให้ access time ของ node ac และ th เปลี่ยนไป คือเป็นเวลาของ www.kmitl.ac.th แทน เมื่อเป็นดังนี้จะเห็นว่า access time จะไม่เรียงลำดับ จากมากไปน้อย จึงมีการปรับโครงสร้างของ Binary ได้ดังรูป



รูปที่ 3.15 binary tree หลังจากการปรับโครงสร้าง

3.1.2.5 Update saved URL

เมื่อต้องการ insert new URL แล้วปรากฏว่ามี URL นี้อยู่ภายใน Binary อยู่แล้วจะต้องทำการ update URL โดยทำการ update access time ให้เป็น access time ของ URL ต้องทำการ insert แทน และทำการปรับโครงสร้างของTreeให้ตรงตามข้อกำหนด

3.1.3 การออกแบบโครงสร้างข้อมูลของ Sequential

โดยที่การเก็บแบบ sequential ก็จะมี

1. Search For URL จะเป็นการ ค้นหา URL ใน cache
2. Insert New URL จะเป็นการเก็บ URL ใน cache เมื่อค้นหาแล้วไม่เจอ URL นั้น
3. Update Saved URL จะเป็นการอัพเดทข้อมูล เมื่อพบ URL ใน cache
4. Delete Saved URL จะเป็นการลบ URL ที่มีความสำคัญน้อยที่สุดใน cacheทิ้งเมื่อ cache เต็ม

3.1.3.1 การเก็บข้อมูลแบบ Sequential โดยใช้ array of structure

เมื่อ structure มีโครงสร้างดังนี้

ชื่อ URL	Pointer to file	Date	Time

รูปที่ 3.16 โครงสร้างของโหนดที่จัดเก็บURLแบบ sequential

โดยที่ - pointer to file คือ pointer ที่ชี้ไปยังที่เก็บไฟล์ใน cache (null is allowed)

- date คือวันที่ทำการ Access page ครั้งล่าสุด
- time คือเวลาที่ทำการ Access page ครั้งล่าสุด

เมื่อเก็บเป็นแบบ sequential ข้อมูลก็จะเรียงกันไปเรื่อยๆ โดยไม่มีการเรียงลำดับความสำคัญของ URL จน cache เต็ม เป็น รูปแบบ array URL

สมมุติว่า cache สามารถเก็บได้ 30 pages ก็จะเก็บข้อมูลดังนี้

0	1	2	.....	27	28	29

รูปที่ 3.17 array ที่ใช้เก็บURL

ในการ Access URL ก็จะเก็บข้อมูลไปเรื่อยๆ โดยไม่มีการเรียงลำดับข้อมูล การเก็บข้อมูลจริง จะเป็นดังนี้

0	www.chula.ac.th	Addr0	31/10/43	7.00
1	www.yahoo.com	Addr1	31/10/43	7.15
2	www.pantip.com	Addr2	3/7/44	8.30
3	www.kmitl.ac.th	Addr3	3/11/43	7.45
4	www.thairath.co.th	Addr4	2/5/44	4.00
..				
..				
28		Null		
29		Null		

รูปที่ 3.18 แสดงการจัดเก็บแบบsequential

Sequential Search การเก็บ records จะเก็บเป็น array เมื่อมี records ก็จะมาต่อท้าย array เมื่อมีการค้นหาข้อมูลก็จะมองทั้ง array

3.1.3.2 Search For URL

Worst Case ของการ ค้นหาข้อมูลแบบ Sequential Big(o) จะเป็น O(n) เพราะต้องค้นหาไปเรื่อยทั้ง array จนกว่าจะเจอข้อมูล ถ้าไม่เจอข้อมูลก็แสดงว่าจะต้องค้นหาจน index สุดท้าย ฉะนั้น จะต้องเปรียบเทียบ (N+1) ตัว โดยเฉลี่ยแล้วการค้นหาจะประสบความสำเร็จก็จะเป็น (N/2) นั่นคือหาข้อมูลไปครึ่ง array แล้วเจอข้อมูล ถ้าเป็น array ที่ข้อมูลมีการเรียงกัน Big(o) ก็จะเป็น (N/2) เช่นกัน

3.1.3.3 Insert New URL

การ insert ข้อมูลจะกระทำก็ต่อเมื่อ ไม่พบ URL ที่ต้องการใน array ฉะนั้นก่อนที่จะกระทำการ insert ได้จะต้องมีการ Search array ก่อนว่ามี URL ที่ต้องการอยู่หรือไม่ การ insert จะมีอยู่ 2 กรณี คือ

- Search แล้วไม่พบ URL ก็แสดงว่า Search มาทั้ง array แล้ว ก็จะ insert URL เข้าไปที่ท้ายสุดของArray (ในกรณีที่ cache ยังไม่เต็ม) ถ้า cache เต็ม ก็จะต้องมาลบ ข้อมูลออกไปก่อนที่จะ insert
- Search แล้วพบ URL ก็จะทำการ update ซึ่งจะกล่าวต่อไป

การ insert URL ใช้ Big(O) = O(n) เท่ากับการ sequential search

3.1.2.4 Update Saved URL

คือการ เก็บข้อมูลของ date และ time ของการ Access ครั้งล่าสุดแทนที่ข้อมูลเก่าใน array เพื่อที่จะนำไปใช้ในการพิจารณาในการลบ URL ออกจาก cache เมื่อ cache เต็ม

3.1.2.5 Delete Saved URL

ใน array จะไม่มีการเรียงข้อมูล ของ date และ time จากมากไปน้อย ฉะนั้น เราต้อง Search ทั้ง array เพื่อที่จะหา minimum date & time Big(o) ของการ delete ก็จะเป็น O(n) เมื่อทำการลบ URL ไปแล้วก็จะ ใส่ URL ใหม่ลงไป สมมุติว่าต้องการลบ 5 URL ซึ่งก็ต้อง search 5 ครั้งลบ 5 ครั้ง ซึ่ง จะเท่ากับ **5n Big(o) ก็จะเป็น O(n)** เท่าเดิมอยู่ดี

3.2 การวิเคราะห์ประสิทธิภาพ

ในหัวข้อนี้จะเป็นการิเคราะห์ประสิทธิภาพของการเก็บ URL ใน 2 รูปแบบคือ แบบ Sequential และแบบ Binary ซึ่งจะทำการวิเคราะห์ในขั้นตอนทั้ง 4 วิธีคือ

1. Search for URL
2. Insert new URL
3. Delete saved URL
4. Update saved URL

3.2.1 แบบ Sequential

สำหรับการเก็บ URL แบบ Sequential นั้นจะทำการจัดเก็บ URL เรียงตามลำดับ URL ที่ทำการเรียกใช้เข้ามาและหากมีการเรียกใช้ URL ที่ซ้ำกับ URL ที่มีอยู่แล้วภายใน URL ที่เก็บใน Array จะทำการแก้ไขที่ access time ของ URL นั้น แต่จะไม่ทำการแก้ไขลำดับของ URLดังนั้นจึงสามารถสรุปได้ดังนี้ คือ

ในกรณี Worst Case - URL ที่ต้องการจะอยู่ท้ายสุดของ Array หรือไม่มีเลย ดังนั้นในการค้นหา URL นั้นจะต้องทำการค้นหา ไปจนถึงสุดของ Array ซึ่งสามารถวิเคราะห์ค่า Big O ออกมาได้เป็น O(n) เช่นเดียวกัน

3.3.2 แบบbinary

ในการเก็บ URLแบบbinary ที่ right link จะเป็นตัวที่ทำการชี้ไปยัง suffix ของURL ต่างๆ และ Left linkจะทำการชี้ไปยัง URL ภายใต้ Suffix เดียวกัน ส่วนภายในsuffixเดียวกัน right linkจะเป็นตัวชี้ไปยังdomain name ภายใต้ suffix เดียวกัน สำหรับกรณี worst case ของbinary คือ จะมีURLภายใต้Suffixเดียวเท่านั้น

ตัวอย่างinputที่เข้ามา เช่น

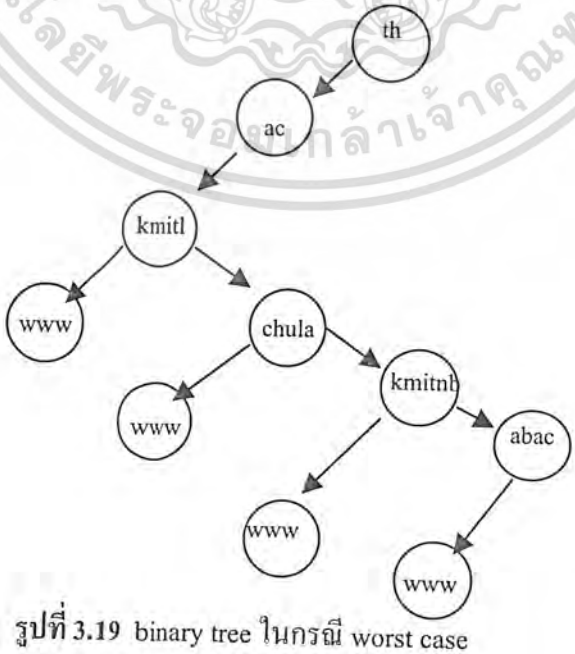
www.kmitl.ac.th

www.chula.ac.th

www.kmitnb.ac.th

www.abac.ac.th

จะได้tree ออกมาดังรูป



รูปที่ 3.19 binary tree ในกรณี worst case

ซึ่งสำหรับกรณี Average Case คือจะมีการกระจายของ Suffix มากๆ ซึ่งจะทำให้ Binary Tree นั้น Balance มากขึ้น ซึ่งถ้าเป็น Balance Tree นั้นค่า Big O จะเท่ากับ $O(\log_2 n)$ แต่สำหรับใน Average Case นั้น เราไม่สามารถสรุปได้ว่ามีค่า Big O ที่แท้จริงนั้นมีค่าเท่าไร ซึ่งจะขึ้นกับ URL ที่เข้ามา แต่สามารถสรุปได้ว่า ค่า Big O จะอยู่ในช่วงของ

** $O(n) \leq$ Big O ของแบบ Binary $\leq O(\log_2 n)$ **

บทที่ 4
การวิเคราะห์และประเมินผลระบบ

จากโครงสร้างที่ใช้จัดเก็บ URL ทั้ง 2 วิธี ที่กล่าวไว้ในบทที่ 3คือ โครงสร้าง sequential และ binary tree ซึ่ง ทั้งสองโครงสร้างจะมีวิธีที่แตกต่างกัน ดังนั้น ในบทที่ 4 นี้จะกล่าวถึงการจัดเก็บ URL ในรูปแบบต่างๆ

4.1 แบบ Sequential

การจัดเก็บ URL แบบ sequential URL ทั้งหมดที่ถูกทำการเรียกใช้ จะถูกจัดเก็บโดยเรียงลำดับตามเวลาที่เรียกใช้(Accesstime) ครั้งแรก โดยไม่สนใจว่าURLจะถูกเรียกใช้อีกหรือไม่ ลำดับของURLที่ทำการจัดเก็บใน cache จะยังคงเป็นลำดับตามเดิม โดยสิ่งที่เปลี่ยนแปลงจะมีเพียง Accesstimeของ URLที่ทำการเรียกใช้ครั้งล่าสุดเท่านั้น

การเก็บข้อมูลแบบ sequential จะมีลักษณะดังนี้

0 www.chula.ac.th	Addr0	31/10/43	7.00
1 www.yahoo.com	Addr1	31/10/43	7.15
2 www.pantip.com	Addr2	3/7/44	8.30
3 www.kmitl.ac.th	Addr3	3/11/43	7.45
4 www.thairath.co.th	Addr4	2/5/44	4.00

รูปที่ 4.1 รูปการจัดเก็บแบบ sequential

เมื่อต้องการเรียกใช้ www.kmitl.ac.th ที่เวลา 8.12 วันที่ 3/9/44 ภายในcache จะมีการเปลี่ยนแปลงดังนี้

0 www.chula.ac.th	Addr0	31/10/43	7.00
1 www.yahoo.com	Addr1	31/10/43	7.15
2 www.pantip.com	Addr2	3/7/44	8.30
3 www.kmitl.ac.th	Addr3	3/9/44	8.12
4 www.thairath.co.th	Addr4	2/5/44	4.00

รูปที่ 4.2 การเปลี่ยนแปลงใน cache เมื่อมีการ update เวลา

จะเห็นว่าสิ่งที่มีการเปลี่ยนแปลงใน cache ก็คือ Accesstime ของ www.kmitl.ac.th (URLที่ทำการเรียกใช้) เท่านั้น ส่วนลำดับของURL ยังคงเหมือนเดิม

สำหรับการค้นหาURL ภายใน cache ของโครงสร้างแบบ sequential นั้นกรณีที่แย่ที่สุด (worst case)สำหรับการค้นหา URL ในโครงสร้างรูปแบบนี้ก็คือ URL ที่ต้องการค้นหาอยู่ลำดับท้ายสุดของcacheทำให้การค้นหาURLที่ต้องการนั้นต้องทำการค้นหาจนถึงลำดับสุดท้ายของ cache

4.2 แบบ Binary Tree

สำหรับการจัดเก็บ URLในรูปแบบนี้จะใช้โครงสร้างของ binary tree มาใช้ในการจัดเก็บ ซึ่งการจัดเก็บจะประกอบด้วย

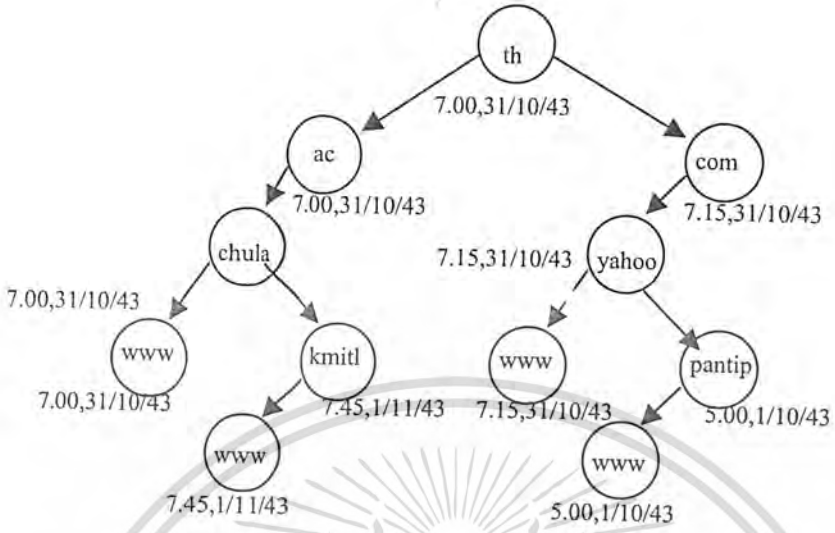
- pointer to left node จะเป็นการเชื่อมต่อระหว่างโหนด ภายในURLเดียวกัน
- pointer to right node จะเป็นการเชื่อมต่อระหว่างโหนด ของURLที่ต่างกัน
- Accesstime จะเป็นเวลาและวันที่ที่ทำการเรียกใช้ URLนั้น

ซึ่งการจัดเก็บในโครงสร้างแบบ binary tree จะมีลักษณะดังนี้

จากตัวอย่างของ Input ที่เข้ามา

www.chula.ac.th	7.00	31/10/43
www.yahoo.com	7.15	31/10/43
www.pantip.com	5.00	1/11/43
www.kmitl.ac.th	7.45	1/11/43

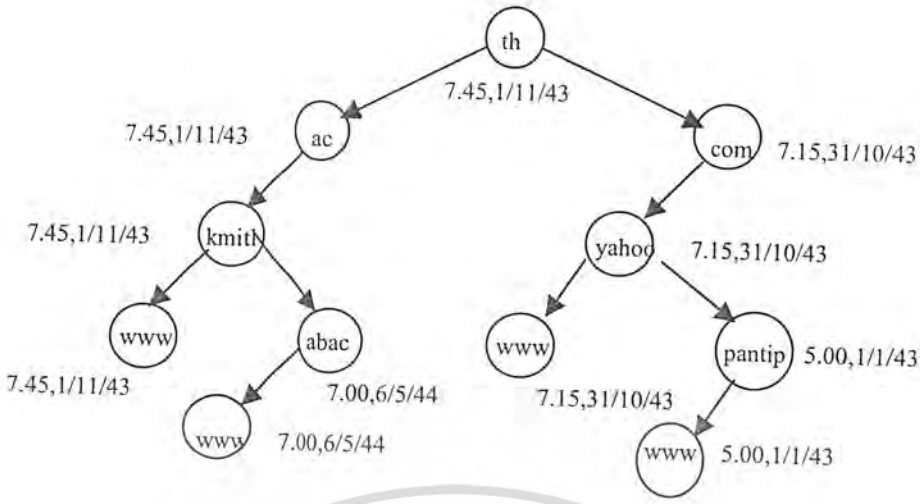
สามารถแปลงให้อยู่ในรูป Binary tree ได้ดังนี้



รูป 4.3 รูปแบบการจัดเก็บ URL แบบ Binary

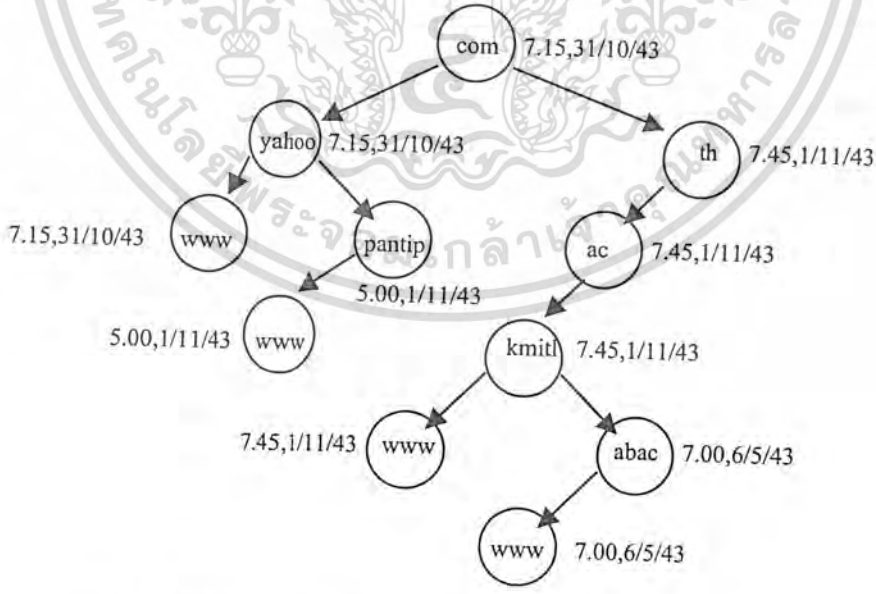
ในการใช้โครงสร้าง แบบ binary tree จะมีการใช้ Accesstime มาใช้ในการจัดลำดับความสำคัญของ URL แต่ละURL ในการจัดเก็บ URL ภายใน โครงสร้าง binary tree ซึ่งเมื่อมีการเปลี่ยนแปลง Accsstime คือมีการเรียกใช้ URL นั้น จะมีการปรับโครงสร้างของ tree ให้อยู่ในรูปแบบที่เหมาะสม ตรงตามข้อกำหนดที่วางไว้ คือณ ตำแหน่ง Root ของแต่ละ URL จะเก็บAccessTime ที่นานที่สุดของURLภายใต้ Suffix เดียวกัน

ตัวอย่างการเปลี่ยนแปลงโครงสร้างของ binary tree เมื่อมีการ เรียกใช้ URL ภายใน cache เมื่อต้องการทำการเรียกใช้ URL ที่ต้องการแต่ปรากฏว่า Cache เต็ม ก็จะต้องทำการหา URL ที่เหมาะสมเพื่อทำการ delete ออกไป สำหรับที่จะนำ URLที่ถูกเรียกใช้มาเก็บใน cache แทน จากตัวอย่าง จะทำการเรียกใช้ www.abac.ac.th แต่ปรากฏว่า cache เต็ม จึงต้องทำการ search หา URL ที่มีเวลาที่ถูกเรียกใช้นานที่สุด ซึ่งในที่นี้ก็คือ www.chula.ac.th จึงได้รูปแบบของ tree ออกมาดังรูป



รูปที่ 4.4 Binary tree หลังจากการ delete www.chula.ac.thและ ทำการ insert www.abac.ac.th

และเมื่อทำการลบ node chula ออกไปจะทำให้ accesstime ของ node ac และ th เปลี่ยนไป คือเป็นเวลาที่ถูกเรียกใช้นานที่สุดในทางสายของ URL ที่มี suffix เดียวกันซึ่งก็คือwww.kmitl.ac.th เมื่อเป็นดังนี้จะเห็นว่า accesstime จะไม่เรียงลำดับ จากมากไปน้อย จึงมีการปรับโครงสร้างของ Binary ได้ดังรูป



รูปที่ 4.5 binary tree หลังจากการปรับโครงสร้าง

จะเห็นว่านอกจากการupdate Accesstime เป็นเวลาครั้งล่าสุดที่ทำการเรียกใช้แล้ว ยังมีการปรับโครงสร้างของTree ให้อยู่ในรูปแบบที่เหมาะสมตรงตามข้อกำหนด

สำหรับรูปแบบการจัดเก็บURL แบบ binary tree ในกรณีที่ แย่ที่สุด (worst case)คือ ในcache จะมี URL ที่มี suffix เดียวกัน สำหรับกรณี worst case ของbinary คือ จะมีURLภายใต้Suffixเดียวเท่านั้น ซึ่งจะทำให้ประสิทธิภาพในการค้นหา เท่ากับ แบบsequential

ตัวอย่างinputที่เข้ามา เช่น

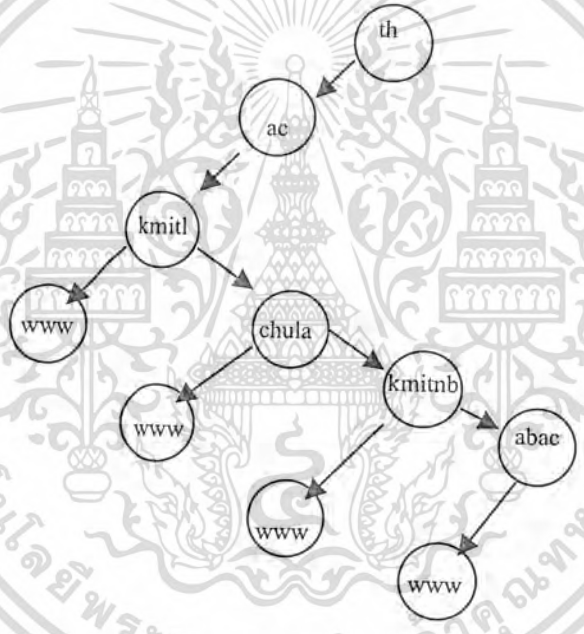
www.kmitl.ac.th

www.chula.ac.th

www.kmitnb.ac.th

www.abac.ac.th

จะได้tree ออกมาดังรูป



รูปที่ 4.5 binary tree ในกรณี worst case

4.3 ข้อเปรียบเทียบระหว่าง โครงสร้าง Sequential และ โครงสร้าง Binary Tree

ในแบบ sequential นั้นการค้นหาURL ที่ต้องการนั้น อย่างน้อยต้องทำการค้นหาภายใน cache แต่หากว่า URL ที่ต้องการอยู่ลำดับสุดท้ายของ cache แล้ว ทำให้การค้นหาครอบคลุมทั้ง cache คือต้องทำการค้นหาตั้งแต่โหนดแรกไปจนถึง โหนดสุดท้าย ซึ่งต่างจากโครงสร้างแบบ binary tree โดยใน โครงสร้าง binary tree จะมีการใช้ pointer to left link และpointer to right link มาช่วยโดย pointer to left link จะเป็นการเชื่อมต่อระหว่างโหนด ภายในURLเดียวกัน

และ pointer to right link จะเป็นการเชื่อมต่อระหว่างโหนด ของURLที่ต่างกัน ทำให้ช่วยลดขอบเขตของการค้นหา ทำให้การค้นหา URLมีจำนวนลดลง

นอกจากนั้นยังมีการใช้ โครงสร้างแบบ binary tree มีการนำ Accesstime มาช่วยใช้ในการจัดลำดับความสำคัญ ทำให้โครงสร้างของการจัดเก็บแบบ binary tree มีความยืดหยุ่นในการใช้งานมากกว่า แบบ sequential

4.4วิธีการคิดประสิทธิภาพของโครงสร้าง Sequential และ โครงสร้าง Binary Tree

4.4.1 แบบ Binary Tree (Average case)

4.4.1.1 Search จะมี 2 กรณี คือ

1. Tree Balance จะได้ Big O = $O(\log_2 n)$
2. Tree not Balance จะได้ Big O ≈ $O(\log_2 n + 4)$

Big O ≈ $O(\log_2 n)$

4.4.1.2 Update ทำการ Search ถ้าพบ URL แล้วจะทำการ Update และจะมีการปรับโครงสร้าง

Big O = O (การ Search + การ Update + การปรับโครงสร้าง)

= O ($\log_2 n$ + 1 + n+2)

≈ O ($\log_2 n$)

4.4.1.3 Insert การ Search แล้วไม่พบ URL จึงทำการ Insert

Big O = O (Search + Insert)

= O ($\log_2 n$ + ค่าของแต่ละสาย)

= O ($\log_2 n$)

4.4.1.4 Delete การ Search ถ้าไม่พบ URL ต้องทำการ Insert แต่ Cache เต็มจึงต้อง Delete URL ทิ้งก่อนที่จะทำการ Insert

Big O = O (Search + Delete + Insert) ทำการ ลบ URL แรกสุดทิ้งและทำการ Sort ในแต่ละ Level

= O ($\log_2 n$ + 1 + $\log_2 n$)

= O ($2\log_2 n + 1 + n$)

= O ($\log_2 n$)

4.4.2 แบบ Sequential (Average Case)

4.4.2.1 Search ถ้า URL ที่ต้องการไม่ได้อยู่ที่ตำแหน่งแรกแ หรือตำแหน่งสุดท้าย จะได้ Big O = O(n)

4.4.2.2 Insert เมื่อ Search แล้วไม่พบ URL แล้วจะทำการ Search แล้ว Insert

Big O = O (การ Search + การ Insert)

= O (n+1) = O (n)

4.2.2.3 Update = (Search + Update)

= O (n+1) เมื่อ Search พบแล้วจึงทำการ Update

≈ O (n)

4.2.2.4 Delete = O (n)

= O (Search + Delete)

= O (n) + O(1)

= O (n) ทำการ Search หาค่า accesstime ที่มีค่านานที่สุด และทำการ Delete

Average Case	Binary Tree	Sequential
Search	$O(\log_2 n)$	O(n)
Insert	$O(\log_2 n)$	O(n)
Update	$O(\log_2 n)$	O(n)
Delete	O(1)	O(n)

ตาราง แสดงค่า Big O ของ โครงสร้างข้อมูล

4.5 การวิเคราะห์ประสิทธิภาพ

สำหรับการวิเคราะห์ประสิทธิภาพของโครงสร้าง Sequential และ โครงสร้าง Binary Tree จะทำการวิเคราะห์โดยค่า Big O โดยในการทำการวิเคราะห์ใน 4 ขั้นตอนคือ

1. Search for URL
2. Insert new URL
3. Update saved URL
4. Delete saved URL

4.5.1 แบบ Sequential

4.4.1.1 Search for URL

จะทำการค้นหา URL ที่อยู่ใน Linklist โดยจะมี Big O เป็นO(n) เพราะว่า การค้น หา URL ใน Linklist ก็จะเป็นการค้นหาแบบ Sequential เพราะฉะนั้นประสิทธิภาพของ จะเท่ากับ ประสิทธิภาพของ Sequential ซึ่งนั้นก็คือ การ Search ใช้ Big O = O(n)

4.5.1.2. Insert new URL

ก่อน insert จะต้องทำการ search ดูก่อนว่ามีอยู่แล้วหรือเปล่า ฉะนั้น จะแบ่งได้เป็น

1. Search ถึงตำแหน่งสุดถ้ายแล้วไม่พบ ก็จะทำการ insert URL ใหม่เข้าไป ณ ตำแหน่งท้าย สุด (กรณีที่ Cache ยังไม่เต็ม)
2. Search แล้วพบ URL ใน Linklist ให้ทำการ Update saved URL

ดังนั้น ประสิทธิภาพของการ Insert จะใช้ Big O = O(n)

4.5.1.3 Update saved URL

เมื่อพบ URL ใน Linklist แล้วก็จะทำการ Update ข้อมูลของการ Access URL ครั้งล่าสุดแทนที่ข้อมูลเก่าของ URL นั้น ใน Linklist ดังนั้น การ Update ใช้ Big O = Big O ของการ Search = O(n)

4.5.1.4 Delete saved URL

จากการที่เราเก็บข้อมูลโดยไม่ได้เรียงตามวันและเวลาจากมากไปน้อย ดังนั้นจึงต้องทำการ search เพื่อทำการค้นหา URLที่เคยถูกเรียกใช้มานานที่สุด (LRU) ซึ่งจะให้การ search หาทั้ง Linklistก่อนแล้วถึงจะรู้ เมื่อลบเสร็จแล้วก็จะเลื่อนอันต่อไปขึ้นมาแล้ว ทำการเพิ่ม URL ใหม่เข้ามา

ดังนั้นการ Delete ใช้ Big O = Big O ของการ Search = O(n)

จากการวิเคราะห์ประสิทธิภาพของการทำงานของรูปแบบ Sequential ทั้ง4 วิธีจึงสามารถสรุปได้ว่าการทำงานแบบ Sequential ใช้ Big O = O(n)

4.5.2 แบบ Binary Tree

4.5.2.1 Search for URL

ประสิทธิภาพของการ Search ใน Binary Tree = $O(\log_2 n)$แต่ Binary Tree ที่ ใช้ในการเก็บ URL ในที่นี้ อาจจะไม่เป็น Complete Binary Tree ก็ได้ ดังนั้นสามารถสรุปได้ว่า

$O(\log_2 n)$ <= ประสิทธิภาพของการ Search ของ Binary Tree ใน โปรเจคนี้< O(n)

4.5.2.2 Insert new URL

ก่อน insert ก็ต้อง search ดูก่อนว่ามีอยู่ใน cachesรือไม่ ฉะนั้นจะแบ่งได้เป็น

1. มี URL อยู่ใน Tree อยู่แล้ว ก็จะทำการ Update
2. ไม่มี URL อยู่ใน Tree ซึ่งจะทำการ Insert

ดังนั้นจึงสามารถสรุปได้ว่า

การ Insert ใช้ Big O = Big O ของการ Search = $O(\log_2 n)$

4.5.2.3 Update saved URL

จะทำการ Update ก็เมื่อมีการ Insert แล้วพบ ชื่อ URL ในแต่ละ Nodeจะทำ การ Update เวลาที่มี Access ครั้งล่าสุด ซึ่ง Big O จะได้มาจาก Big O ของ Search + Update ซึ่งก็เท่ากับ $O(\log_2 n)$ดังนั้นจึงสามารถสรุปได้ว่า

การ Update ใช้ Big O = Big O ของการ Search = $O(\log_2 n)$

4.5.2.4 Delete saved URL

เมื่อเกิดการ Delete ก็จะทำการลบ Node สายแรกออกไป แล้วจะเกิดการย้าย Tree ให้สายแรกเป็น URL ที่ถูกเรียกใช้มานานที่สุดในcacheโดยที่ Big O จะเกิดจาก Search แล้วไม่พบจึงต้อง Insert เข้าไป แต่ปรากฏว่า cache เต็มจึงต้องทำการ delete เพราะฉะนั้น Bio O จะเป็น $O(\log_2 n)$ ดังนั้นจึงสามารถสรุปได้ว่า

การ Delete ใช้ Big O = Big O ของการ Search = $O(\log_2 n)$

จากการวิเคราะห์ประสิทธิภาพของการทำงานของรูปแบบ Binary tree ทั้ง4 วิธีจึงสามารถสรุปได้ว่าการทำงานแบบ Binary Tree ใช้ Big O = O(n)

เมื่อทำการวิเคราะห์ประสิทธิภาพของโครงสร้างทั้งสองรูปแบบทำให้สามารถสรุปได้ว่า

$O(\log_2 n)$ <= Big Oของ Binary Treeในโปรเจคนี้ < O(n)

บทที่ 5
บทสรุปและข้อเสนอแนะ

สรุปผล

ผลที่ได้จากการได้ใส่ URL ตัวอย่างเข้าไปในโปรแกรมทั้ง 2 ตัว ทำให้เราทราบว่าการทำงานในส่วน backend นั้นมีการทำงานอย่างไร ซึ่งการทำงานของโปรแกรมจะเป็นการจำลองการทำงานเท่านั้น ไม่สามารถเอาไปใช้ในการทำงานเก็บไฟล์ HTML จริง ๆ เพื่อใช้ในการแสดงผลใน Browser ได้ซึ่งทางผู้จัดทำได้ทำโปรแกรมในการแสดงผลแยกออกจากกับเป็น 3 ส่วน

- โปรแกรมจำลองการทำงานของการเก็บ Cache แบบ Sequence
- โปรแกรมจำลองการทำงานของการเก็บ Cache แบบ Binary Tree
- โปรแกรมจำลองการสร้างไฟล์ Cache เพื่อใช้เป็นตัวอย่างในการทดลอง

โดยจะมีข้อจำกัดของระบบ คือ โปรแกรมนี้ต้องทำงานบน Windows98 กับสามารถรับ URL ที่เข้ามาใช้ในระบบได้เพียง URL ที่เป็น root ของแต่และ URL เช่น

www.kmitl.ac.th

www.yahoo.com

ไม่สามารถรับ URL ที่เป็นแบบ www.kmitl./science/s00560xx/index.html อย่างนี้ได้

สรุปประสิทธิภาพของอัลกอริทึมแบบต่าง ๆ

- **แบบ Sequential**

สำหรับการเก็บ URL แบบ Sequential นั้นจะทำการจัดเก็บ URL เรียงตามลำดับ URL ที่ทำการเรียกใช้เข้ามาและหากมีการเรียกใช้ URL ที่ซ้ำกับ URL ที่มีอยู่แล้วภายใน URL ที่เก็บใน Array จะทำการแก้ไขที่ Access Time ของ URL นั้น แต่จะไม่ทำการแก้ไขลำดับของ URL ดังนั้นจึงสามารถสรุปได้ดังนี้ คือ

ในกรณี Worst Case URL ที่ต้องการจะอยู่ท้ายสุดของ Array หรือไม่มีเลย ดังนั้นในการค้นหา URL นั้นจะต้องทำการค้นหา ไปจนถึงสุดของ Array ซึ่งสามารถวิเคราะห์ค่า Big O ออกมาได้เป็น O(n) เช่นเดียวกัน

● **แบบ Binary Tree**

ในการเก็บ URLแบบbinary ที่ right link จะเป็นตัวที่ทำการชี้ไปยัง suffix ของURL ต่างๆ และ Left linkจะทำการชี้ไปยัง URL ภายใต้ Suffix เดียวกัน ส่วนภายในsuffixเดียวกันright linkจะเป็นตัวชี้ไปยังdomain name ภายใต้ suffix เดียวกัน

สำหรับกรณี worst case ของbinary คือ จะมีURLภายใต้Suffixเดียวเท่านั้น ซึ่งสำหรับกรณี Average Case คือจะมีการกระจายของ Suffix มากๆ ซึ่งจะทำให้ Binary Tree นั้น Balance มากขึ้น ซึ่งถ้าเป็น Balance Tree นั้นค่า Big O จะเท่ากับ $O(\log_2 n)$ แต่สำหรับใน Average Case นั้น เราไม่สามารถสรุปได้ว่ามีค่า Big O ที่แท้จริงนั้นมีค่าเท่าไร ซึ่งจะขึ้นกับ URL ที่เข้ามา แต่สามารถสรุปได้ว่า ค่า Big O จะอยู่ในช่วงของ

$$O(n) \geq \text{Big O ของแบบ Binary} \geq O(\log_2 n)$$

ข้อเสนอแนะ

1. ควรที่จะสามารถนำไปประยุกต์ใช้งานได้จริงกับ Browser ต่าง ๆ ได้
2. เนื่องจากโปรแกรมนี้ได้ถูกออกแบบโดยการเขียนโปรแกรมในเชิงวัตถุ (OOP : Object Oriented Programming) ดังนั้นจึงนำคลาส (CLASS) ซึ่งออกแบบไว้ไปพัฒนาต่อ เพื่อประยุกต์กับระบบที่จะถูกพัฒนาต่อไป
3. การที่จะทำการพัฒนาโปรแกรมนี้ให้ใช้ได้จริง ผู้ที่จะพัฒนาควรที่จะต้องเน้นการศึกษาเกี่ยวกับเรื่อง Data Structure และ Win API เพื่อที่ใช้ติดต่อกับโปรแกรมต่าง ๆ

ภาคผนวก

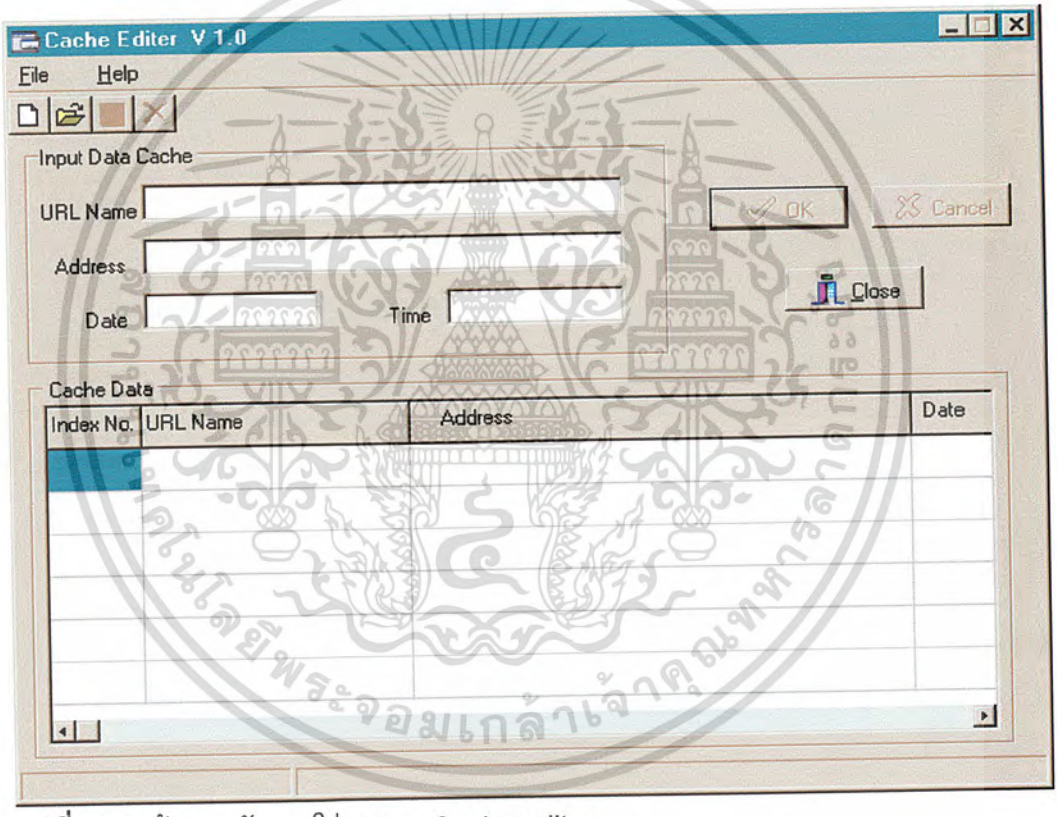
ตัวอย่างชุดภาพ

ตัวอย่างชุดภาพ

ในบทนี้จะกล่าวถึงการดำเนินการของโปรแกรม ซึ่งจะอธิบายเป็นขั้นตอน ต่อๆไป_ในโปรแกรมจะแบ่งออกเป็น 3 ส่วน คือ

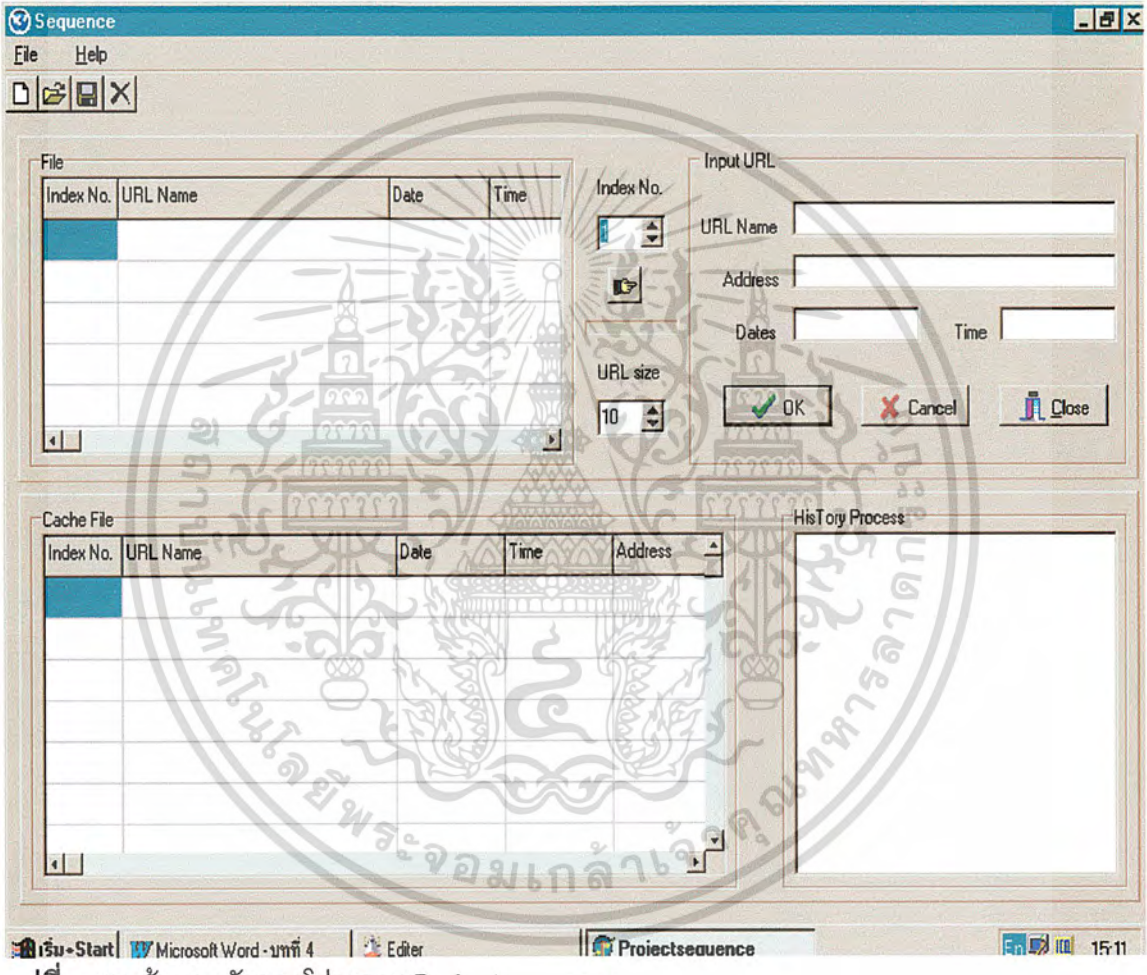
1. Cache editor ซึ่งจะเป็นส่วนที่รับ URL เพื่อมาเก็บไว้ใน Cache เพื่อที่จะได้นำไปใช้ค้นหาในครั้งต่อๆ ไป

   ซึ่งตัวอย่างของโปรแกรมแสดงได้ดังรูป



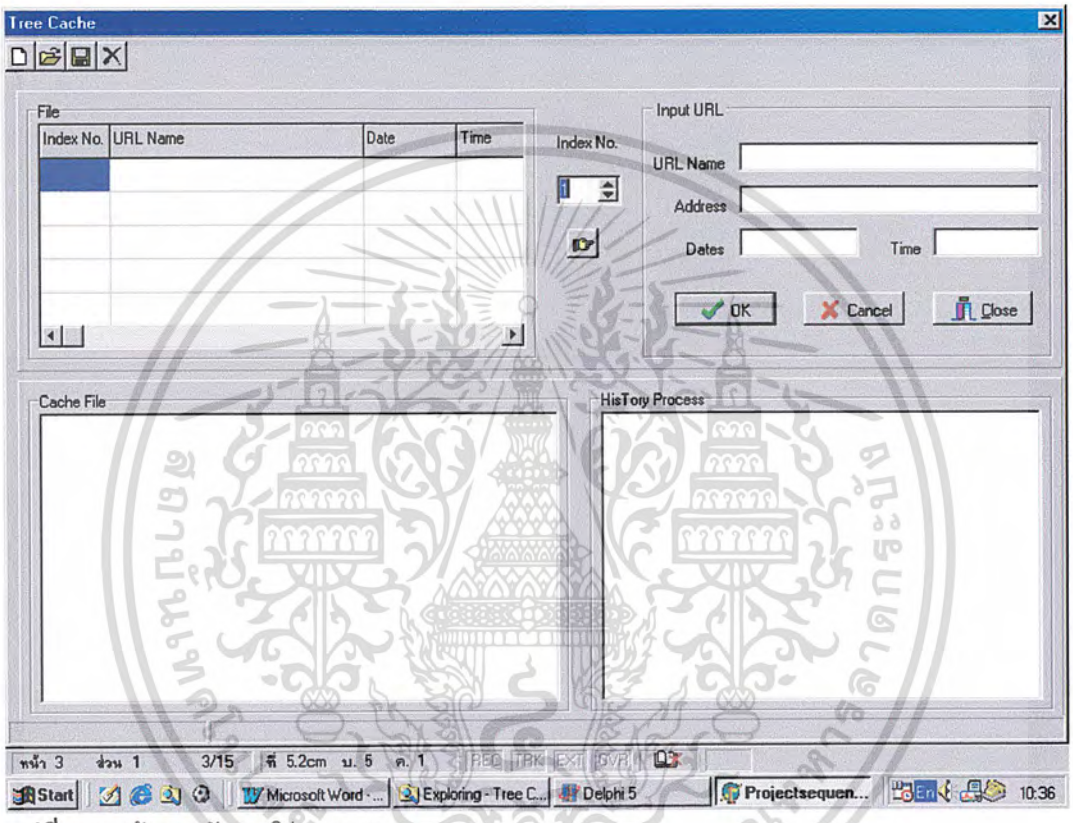
รูปที่ ก-1 หน้าจอหลักของโปรแกรม Cache editor

2. Project sequence ซึ่งเป็นส่วนที่รับ file URL และนำมาจัดเก็บลง Cache พร้อมทั้งมีการ แสดงHistory process เพื่อแสดงการจัดเก็บ ซึ่งโปรแกรมในส่วน sequence จะแสดงได้ดังรูป



รูปที่ ก-2 หน้าจอหลักของโปรแกรม Projectsequence

3. Project Binary Tree ซึ่งเป็นส่วนที่รับ file URL และนำมาจัดเก็บลง Cache พร้อมทั้งมีการแสดงHistory process เพื่อแสดงการจัดเก็บ ซึ่งโปรแกรมในส่วน Tree จะแสดงได้ดังรูป



รูปที่ ก-3 หน้าจอหลักของโปรแกรม Tree

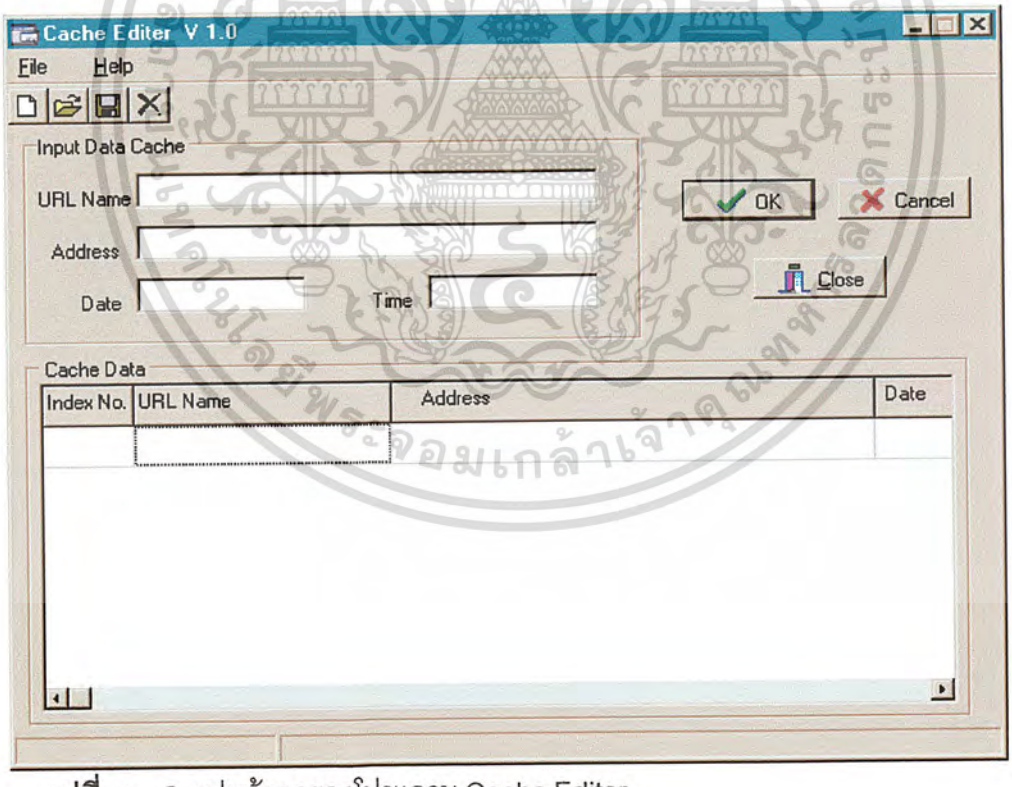
ขั้นตอนการทำงานของโปรแกรม

ในการที่จะค้นหา URL ให้รวดเร็วที่สุดนั้น ก่อนอื่นจำเป็นจะต้องมี URL อยู่ใน Cache เสียก่อนซึ่ง การที่จะนำ URL ลง Cache นั้นจะมีขั้นตอนดังนี้

1. จะต้องใช้โปรแกรม ในส่วน Cache editor เพื่อที่จะนำข้อมูลของ URL นั้นๆ มาเก็บเป็น file ซึ่ง file จะถูกจัดเก็บเป็นสกุล *.cac
2. เมื่อได้ file ที่เป็น .cac มาแล้ว ก็จะใช้โปรแกรม Projectsequence เพื่อที่จะทำการจัดเก็บ URL ลง Cache ที่เราได้จำลองขึ้นมา

ในส่วนของCache editor

เมื่อเริ่มเปิดโปรแกรมขึ้นมาโปรแกรมจะปรากฏดังรูป



รูปที่ ก – 3 รูปหน้าจอของโปรแกรม Cache Editor

จะมี ฟิลด์เพื่อรับค่าต่างๆ ดังนี้

* URL Name คือ ชื่อของ URL ที่ต้องการจะเก็บ

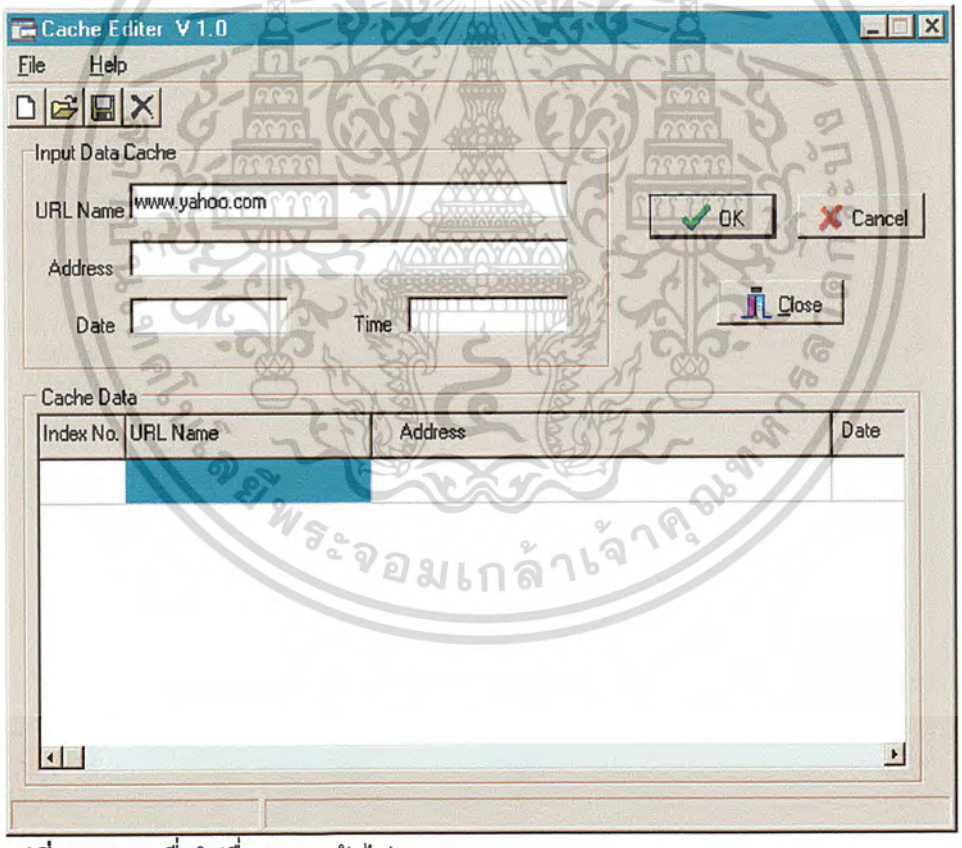
* Address เป็นที่อยู่ของ URL นั้นๆ ใน Cache ที่ทำการจำลองขึ้น

* Date เป็นวันที่เก็บ URL (วัน/เดือน/ปี)

* Time เป็นเวลาที่เก็บ URL ขณะนั้น

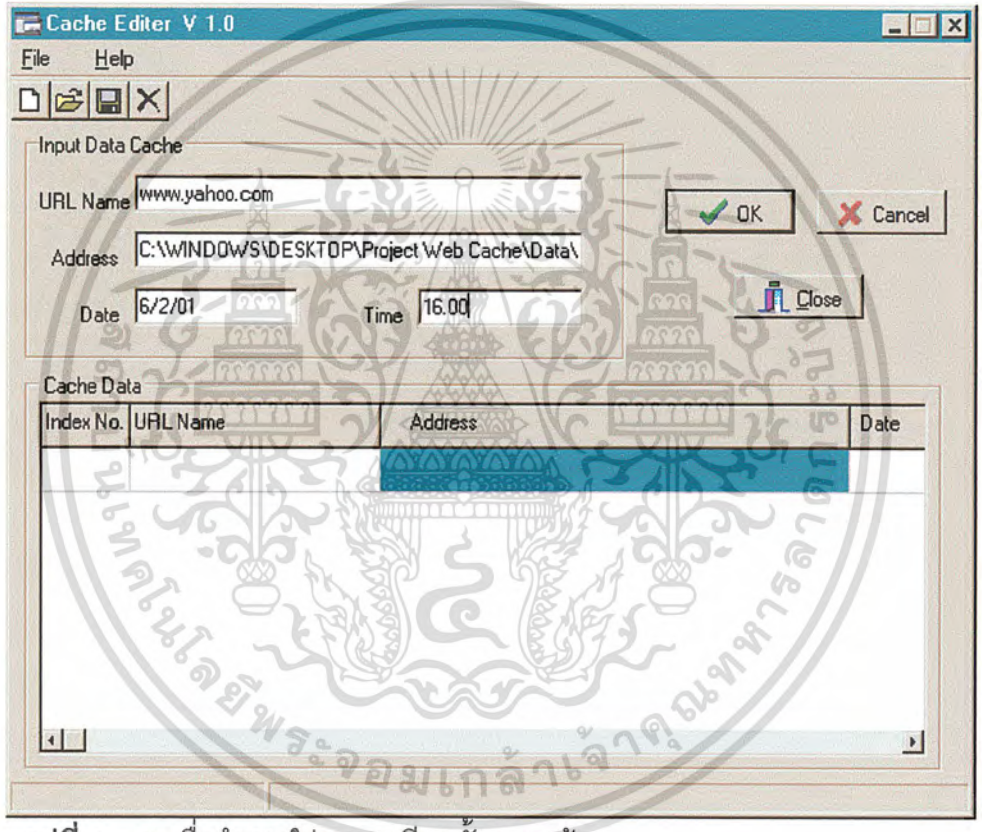
วิธีการใช้โปรแกรม

1. user ทำการใส่ค่า URL Name ซึ่งเป็นชื่อของ URL ที่ต้องการจัดเก็บลง Cache



รูปที่ ก – 4 เมื่อใส่ชื่อ URL เข้าไป

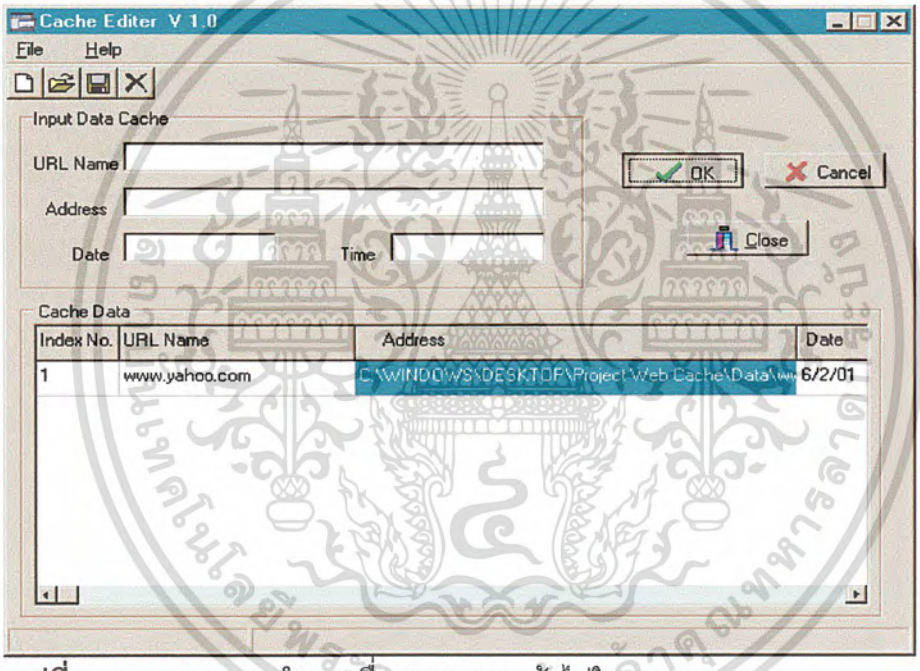
2. เมื่อ user คลิ้ก เม้าส์ ไปที่ ฟิลด์ Address ที่อยู่ของ URL ที่จะอยู่ใน Cache จำลองซึ่งจะอยู่ในเครื่องคอมพิวเตอร์ จะปรากฏขึ้นมาโดยอัตโนมัติ และทำการใส่วันที่และเวลาที่เรียกใช้



รูปที่ ก – 5 เมื่อทำการใส่รายละเอียดทั้งหมดแล้ว

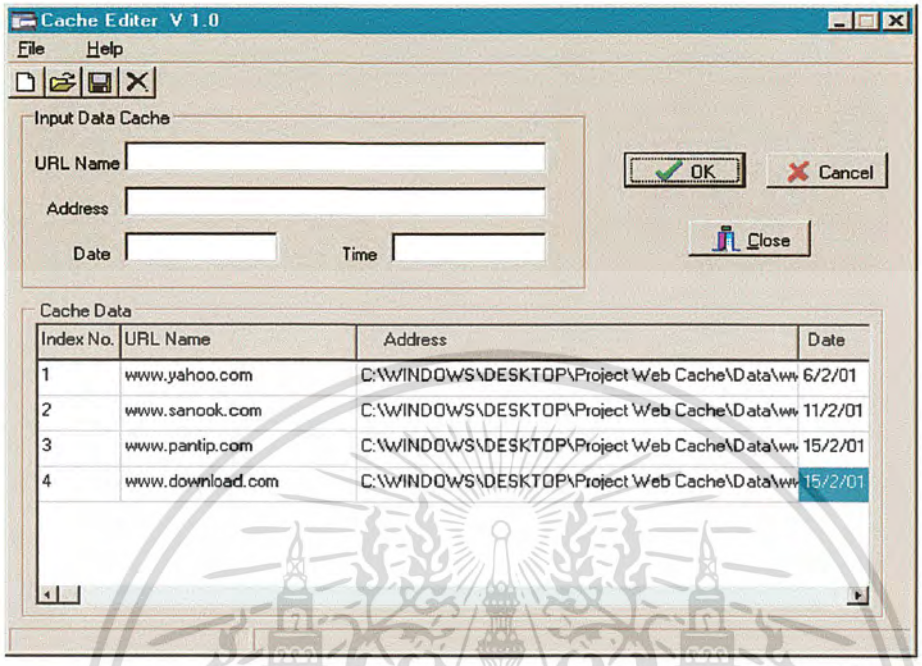
4. กด ok เมื่อต้องการที่จะบันทึกข้อมูลของ URL ลงใน table หรือกด cancle เมื่อต้องการจะกรอกข้อมูล ใหม่ เมื่อเกิดความผิดพลาด

5. กดปุ่ม ok เพื่อบันทึกข้อมูล เมื่อกดปุ่ม ok โปรแกรมจะแสดงได้ดังรูป



รูปที่ ก – 5 แสดงการทำงานเมื่อ Add URL เข้าไปใน Cache

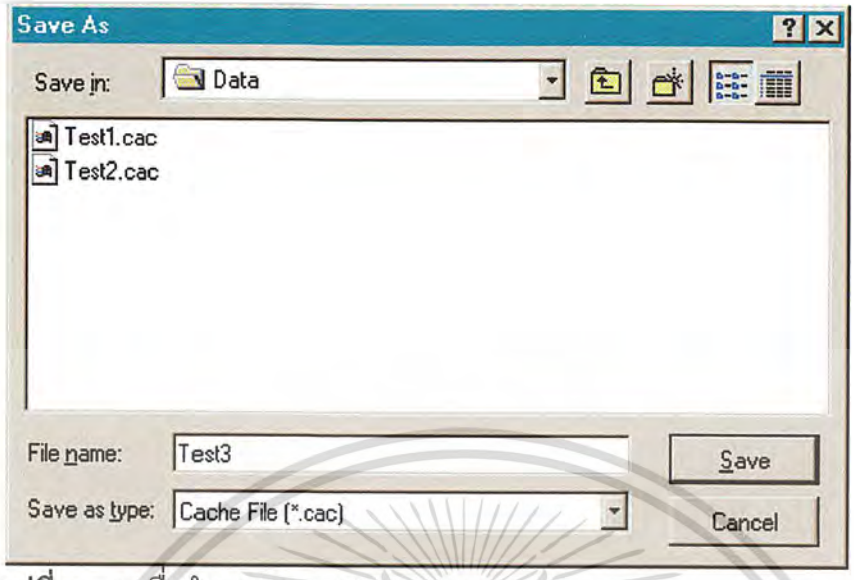
ในกรณีนี้จะใส่ URL 4 ตัว ลง table เพื่อที่ง่ายต่อการอธิบายในส่วนของโปรแกรม



Projectsequence

รูปที่ ก – 6 แสดงรายการ URL ที่มีอยู่ใน Cache

6. จากนั้น ทำการ save file เพื่อจะนำ file นี้ไปเก็บลง Cache จริงๆ ซึ่ง file ที่ได้จะเป็น file *.cac



รูปที่ ก – 7 เมื่อทำการ Save File ของ Cache

ในที่นี้จะตั้งชื่อ file เป็น Test3 จากนั้นก็ save file ในขั้นตอนนี้เราจะได้ file *. Cac เพื่อที่จะไปใช้ในโปรแกรมส่วนที่ 2 คือ Projectsequence แล้ว

ในส่วนของโปรแกรม Projectsequence

โปรแกรมนี้จะใช้ต่อเนื่องจาก โปรแกรม Cache editor ที่ได้ file .cac เพื่อที่จะนำมาจัดเก็บในCache รวมทั้งยังมีการ insert ,update และ delete URL ใน Cache ด้วย หน้าจอหลักของโปรแกรมคือ รูปที่ 4.2

การรับ file ที่มาจากโปรแกรม Cache editor เพื่อนำมาเก็บใน Cache ซึ่งในโปรแกรมนี้จะแบ่งออกได้เป็น 4 ส่วน

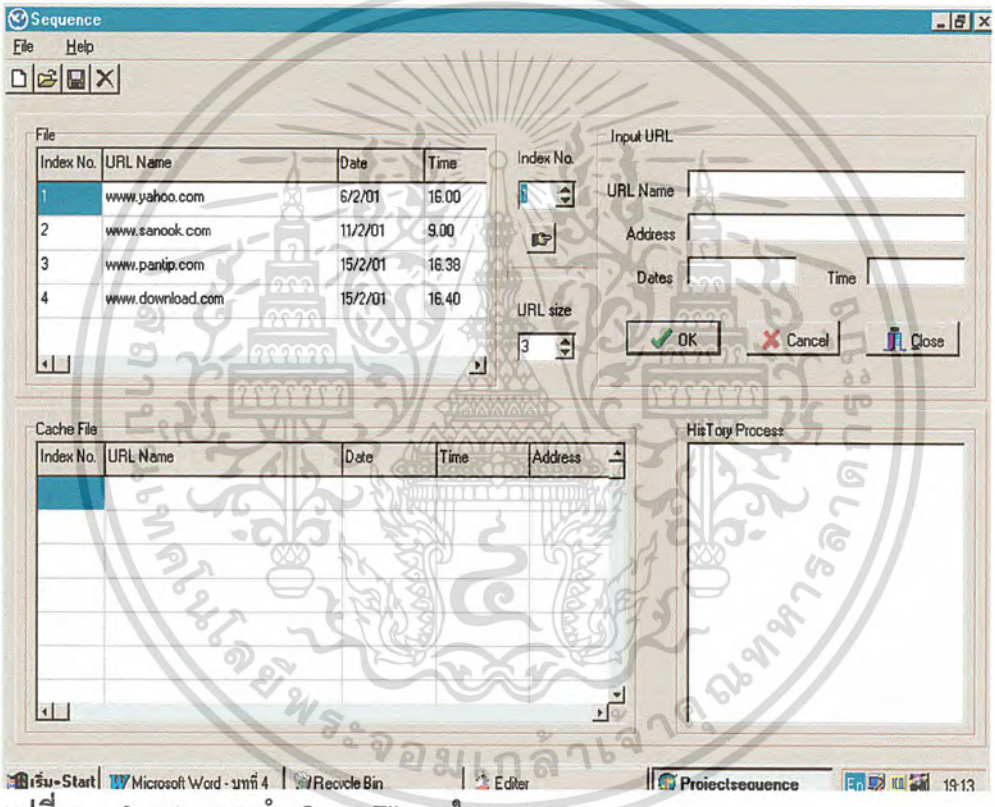
1. file เป็น table ที่นำค่า file *.cac ที่ได้มาจากโปรแกรม Cache editor
2. input URL เป็นส่วนที่ให้ผู้ใช้สามารถเลือกได้ว่าต้องการที่จะนำ URL ใด ไปเก็บใน Cache ซึ่งจะมี index number เพื่อที่จะสามารถเลือก index นั้น เก็บลง Cache และมี URL size เพื่อที่จะกำหนดขนาดของ Cacheว่า สามารถที่จะเก็บ URL ได้กี่ค่า
3. Cache file เป็น table ส่วนที่แสดงการเก็บค่า URL ลงใน Cache จริงๆ
4. History Process เป็นส่วนแสดงการทำงานของ โปรแกรม ในการจัดการ URL ใน Cache

การ InsertURL

เป็นการนำค่าจากโปรแกรม Cache editor เพื่อที่จะจัดเก็บลง Cache ซึ่งในครั้งแรกของ Cache จะยังไม่มี URL อยู่ใน Cache เลย ฉนั้นจึงเป็นการ insert new URL แต่ถ้า insert URL ที่มีอยู่แล้ว ก็จะต้อง update URL นั้นๆ ด้วย และถ้า insert URL ไปแล้วแต่ cache เต็มแล้ว ก็จะทำการ delete ซึ่งจะกล่าวต่อไป

วิธีการทำงาน

1.ในขั้นตอนแรก ผู้ใช้ต้อง open file จาก directory ที่เก็บ file *.cac เสียก่อน ในที่นี้จะเรียกไฟล์ Test3.cac ขึ้นมาจัดเก็บใน Cache ซึ่ง file ที่เรียกออกมานั้นจะไปอยู่ในส่วนของ file ซึ่งจะแสดงได้ดังรูปต่อไปนี้ซึ่งURL ในไฟล์ Test3.cac จะมีทั้งหมดอยู่ 4 ค่า



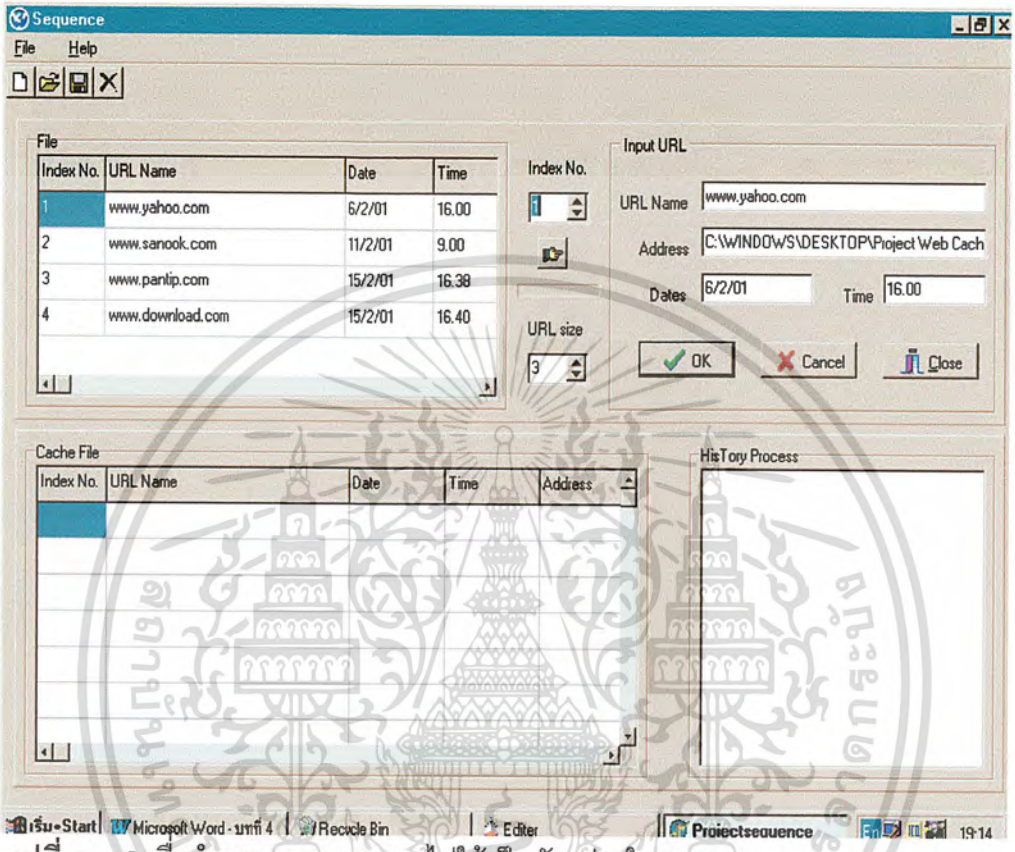
รูปที่ ก – 8 แสดงการทำ Open File ลงในตาราง

2. เมื่อได้ table ในส่วน file แล้ว ผู้ใช้ก็จะสามารถเลือกค่า URL ที่จะจัดเก็บลงใน Cache ได้

โดยเลือกได้จาก index number และในขณะนี้ กำหนดขนาด URL size ไว้ที่ 3 นั่นคือ Cache สามารถเก็บ

ค่า URL ได้ 3 ค่า ซึ่งจะเป็น new URL ทั้งหมดเพราะยังไม่มี URL อยู่ใน Cache เลย

3. เลือกค่า index number ในที่นี้จะเลือก index number 1 มาจัดเก็บ
4. กดปุ่มรูป มือ เพื่อที่จะนำค่าข้อมูล URL ไปไว้ในส่วนของ input file แสดงได้ดังรูป

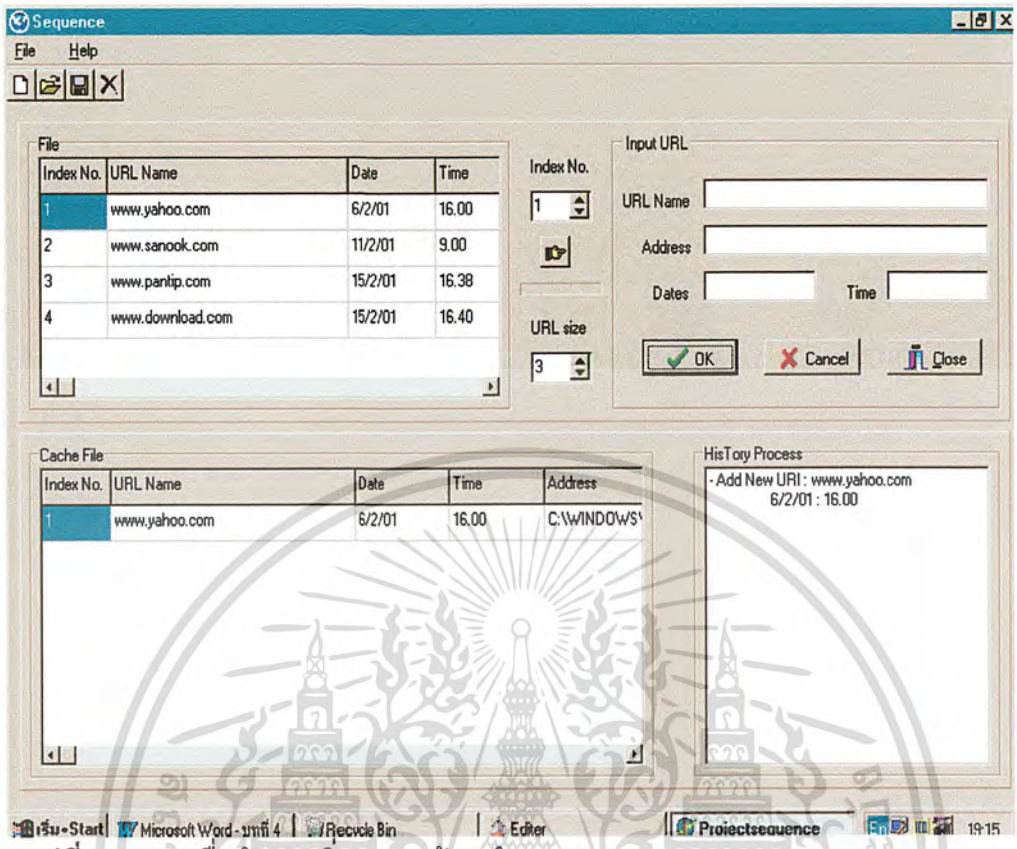


รูปที่ ก – 9 เมื่อนำรายการจากตารางไปใช้เป็นตัวอย่างในCache

จะนำเอาข้อมูลของ www.yahoo.com ซึ่งมี index number เท่ากับ 1 มาไว้ในส่วน input file และกดปุ่ม ok

ถ้าตกลงจะนำ URL นี้เก็บลง Cache และสามารถยกเลิกการ insert ได้เมื่อไม่ต้องการ ค่า URL นี้ โดยกด ปุ่ม cancle

5. เมื่อกดปุ่ม ok แล้ว ค่าข้อมูลของ URL ก็จะถูกเก็บใน table ของ Cache fileและในส่วนของ History Process ก็จะแสดงการทำงานด้วย

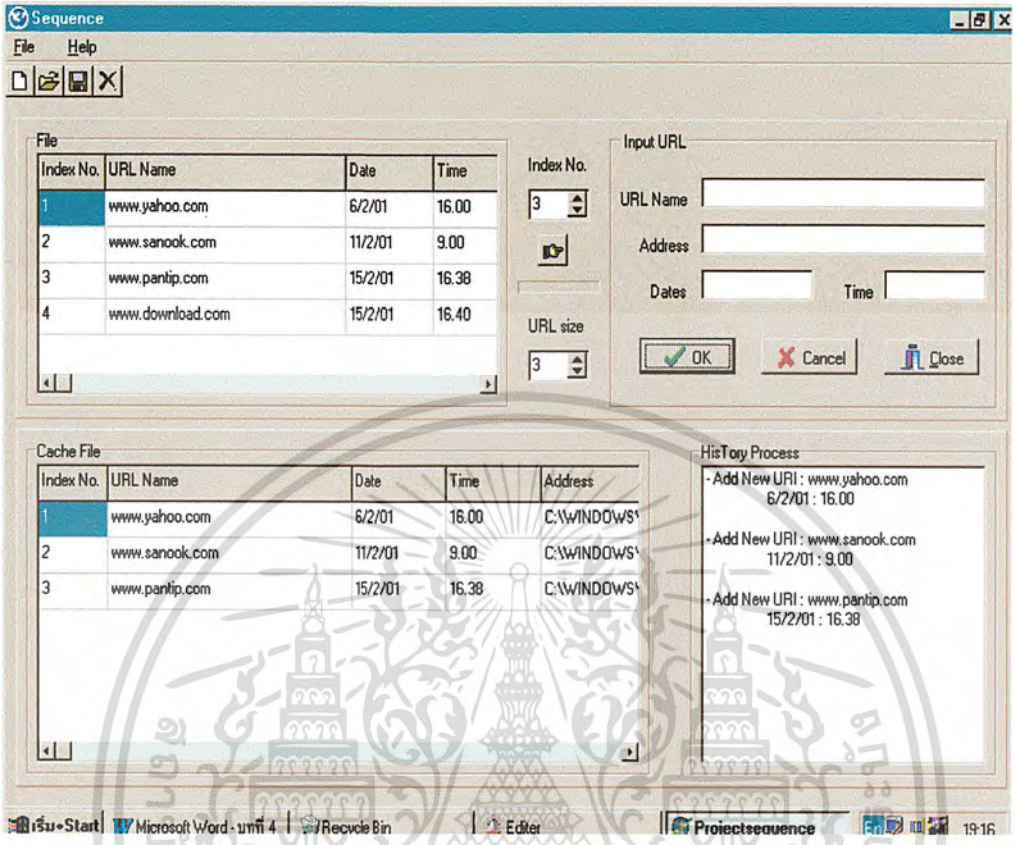


รูปที่ ก – 10 เมื่อทำการเพิ่ม URL เข้ามาใน Cache

ส่วนของ History Process จะแสดงการทำงานพร้อมทั้งแสดงวัน และ เวลาที่ insert URL Process ขณะ insert www.yahoo.com ก็คือ Add NewURL เพราะยังไม่มี www.yahoo.com ใน Cache มาก่อนเลย

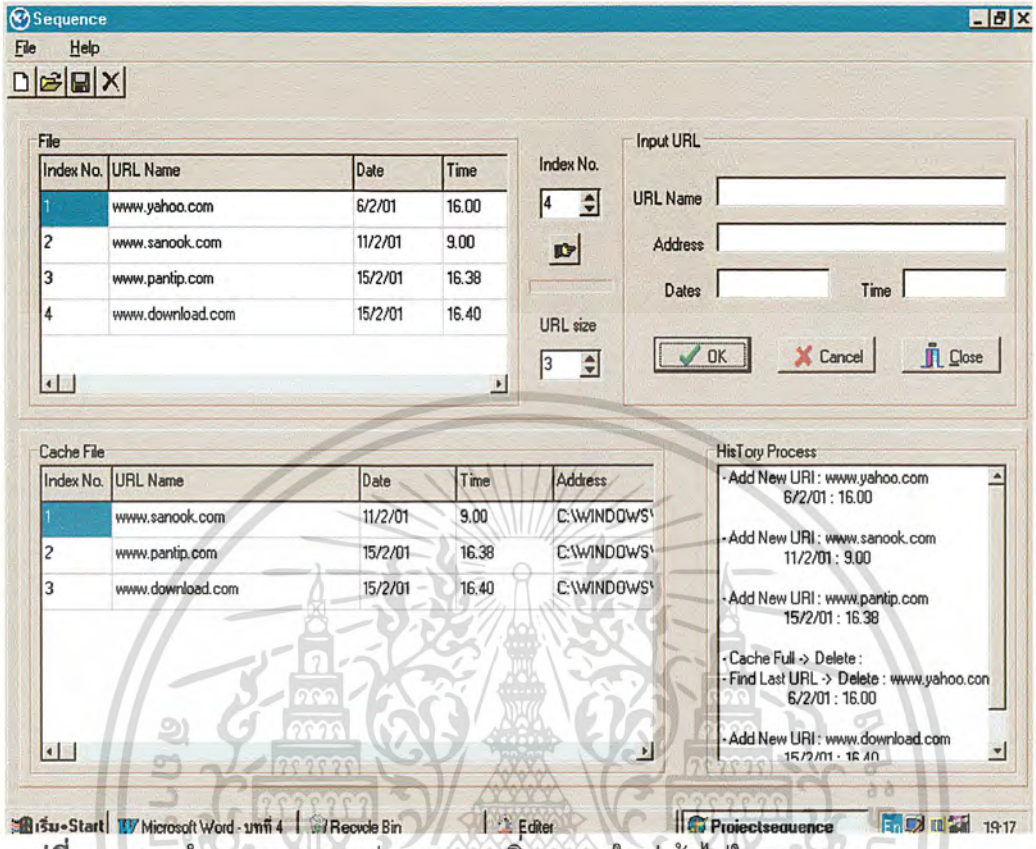
การ DeleteURL

ในหัวข้อ insert URL จะทำการ insert ทั้งไฟล์ Test3.cac ซึ่งมีค่า URL อยู่ 4ค่า แต่ขนาดURL size กำหนดไว้ที่ 3 ฉนั้นในการ insert ตัวที่ 4 ลงไปใน Cache และค่า URL ตัวที่ 4 ไม่ซ้ำกับ URL ใน cache จึงต้อง delete URL ที่ไม่ได้เรียกใช้มานานออกไป ซึ่งจะแสดงให้เห็นดังรูป



รูป ก – 11 เมื่อมีการเพิ่ม URL จนเต็มแล้ว

จะเห็นว่ายังไม่ต้องลบเพราะยังไม่เกิน ขนาดของURL size นั่นคือ Cache ยังไม่เต็มนั่นเอง รูปต่อไปจะแสดงการ insert เมื่อ cache เต็มแล้ว นั่นคือ insert URL ที่ 4 เข้ามา



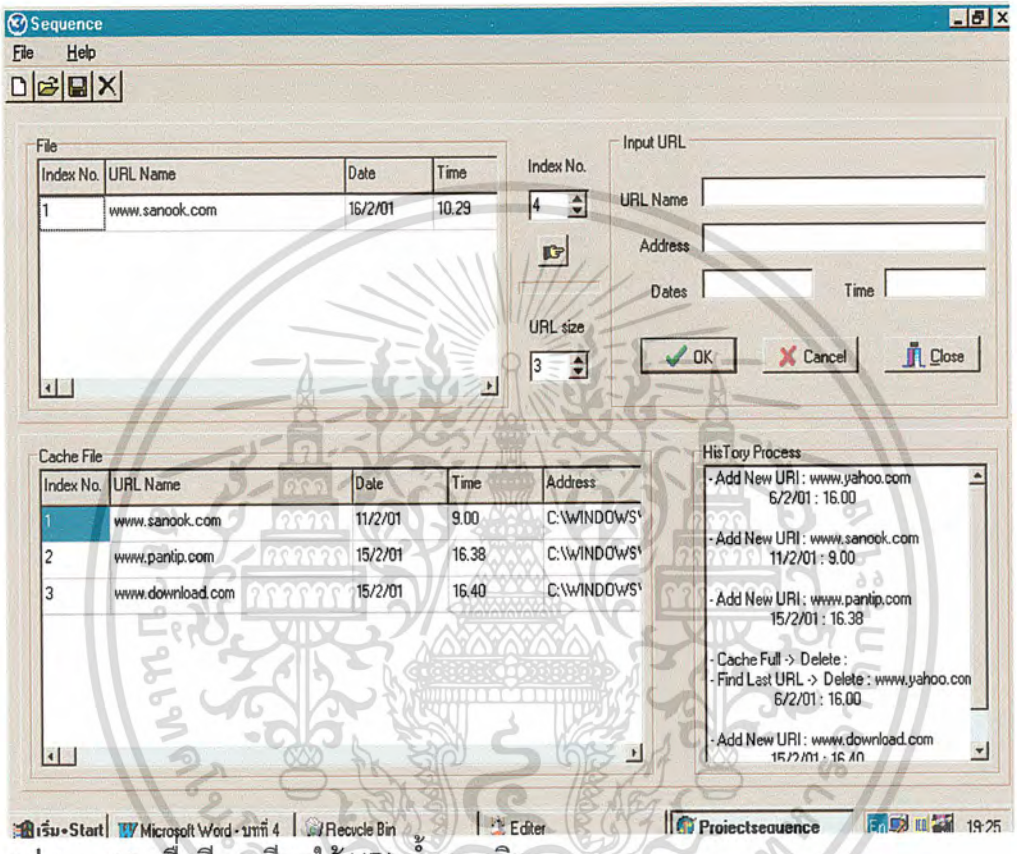
รูปที่ ก – 12 ทำการลบ URL เก่าออกและเพิ่ม URL ใหม่เข้าไปใน Cache

เมื่อ insert URL ใหม่เข้ามา ซึ่งเป็น www.download.com History Process จะแสดงออกมาทางหน้าจอว่า Cache full และทำการ delete URL ที่ไม่ได้ถูกเรียกใช้นานที่สุด ในที่นี้ก็จะเป็น www.yahoo.com เพราะไม่ได้ถูกเรียกใช้นานที่สุด ซึ่งดูจาก วัน และเวลาของ URL นั้น และค่าใน table ของ Cache file ก็เปลี่ยนไปด้วย โดย www.yahoo.com หายไป และ www.download.com
เข้ามาแทน

การ UpdateURL

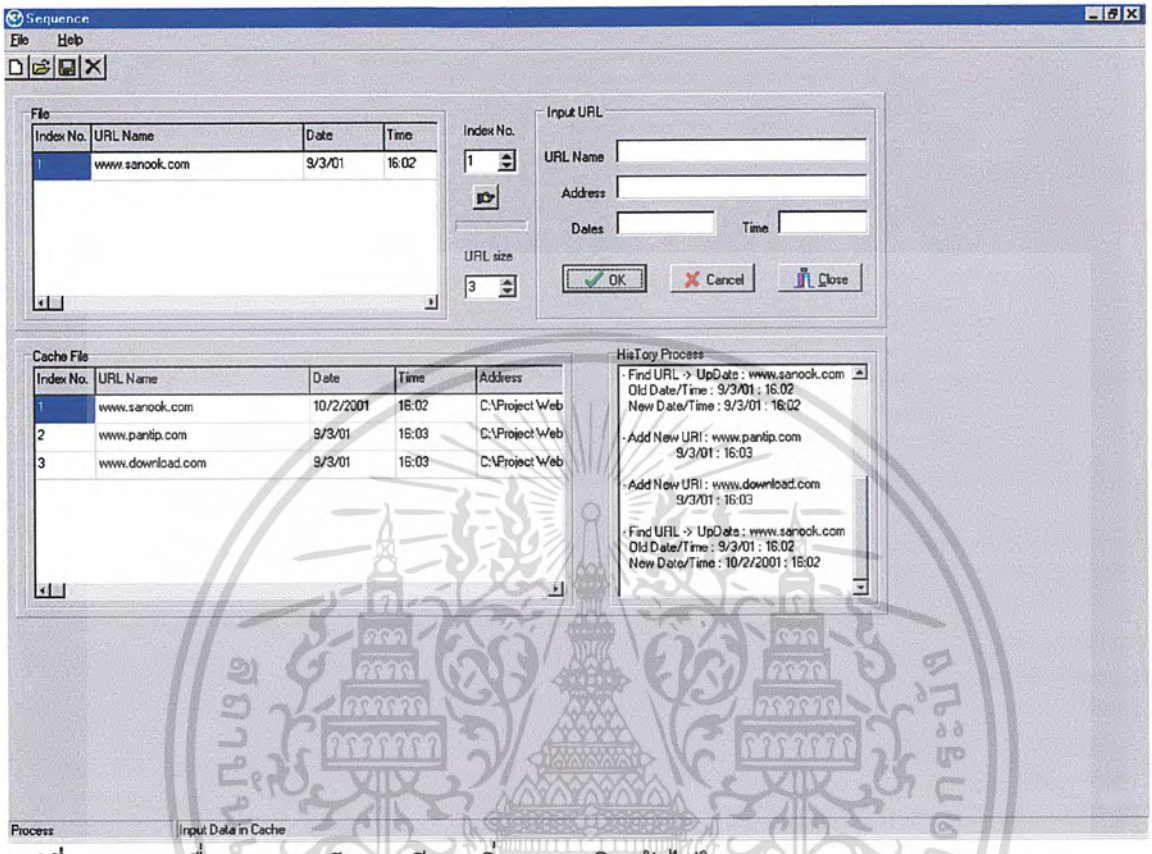
เกิดขึ้นในกรณีที่ ค่า URL ใหม่ที่เข้ามานั้นซ้ำกับ ค่า URL เดิม โดยจะทำการ update วันและเวลาที่เข้ามาใน Cache ของ URL นั้น เพื่อที่จะได้ไม่มีปัญหาในการลบ URL ออกจาก Cache จากรูป 4.14 ถ้าเกิดมี

file Test4.cac เข้ามา ซึ่ง file นี้เก็บค่า URL ของ www.sanook.com ซึ่งแสดงได้ดังรูป



รูป ก - 13 เมื่อมีการเรียกใช้ URL ซ้ำของเดิม

และเราต้องการ insert URL นี้ลง Cache Cache ก็จะทำการ update URL ดังรูป ก -14



รูปที่ ก – 14 เมื่อ Cache เต็มและมีการเพิ่ม URL เดิมเข้าไปใน Cache

ในส่วนของโปรแกรม Project Tree

โปรแกรมนี้จะใช้ต่อเนื่องจาก โปรแกรม Cache editor ที่ได้ file .cac เพื่อที่จะนำมาจัดเก็บใน

Cache รวมทั้งยังมีการ insert ,update และ delete URL ใน Cache ด้วย หน้าจอหลักของโปรแกรมคือ

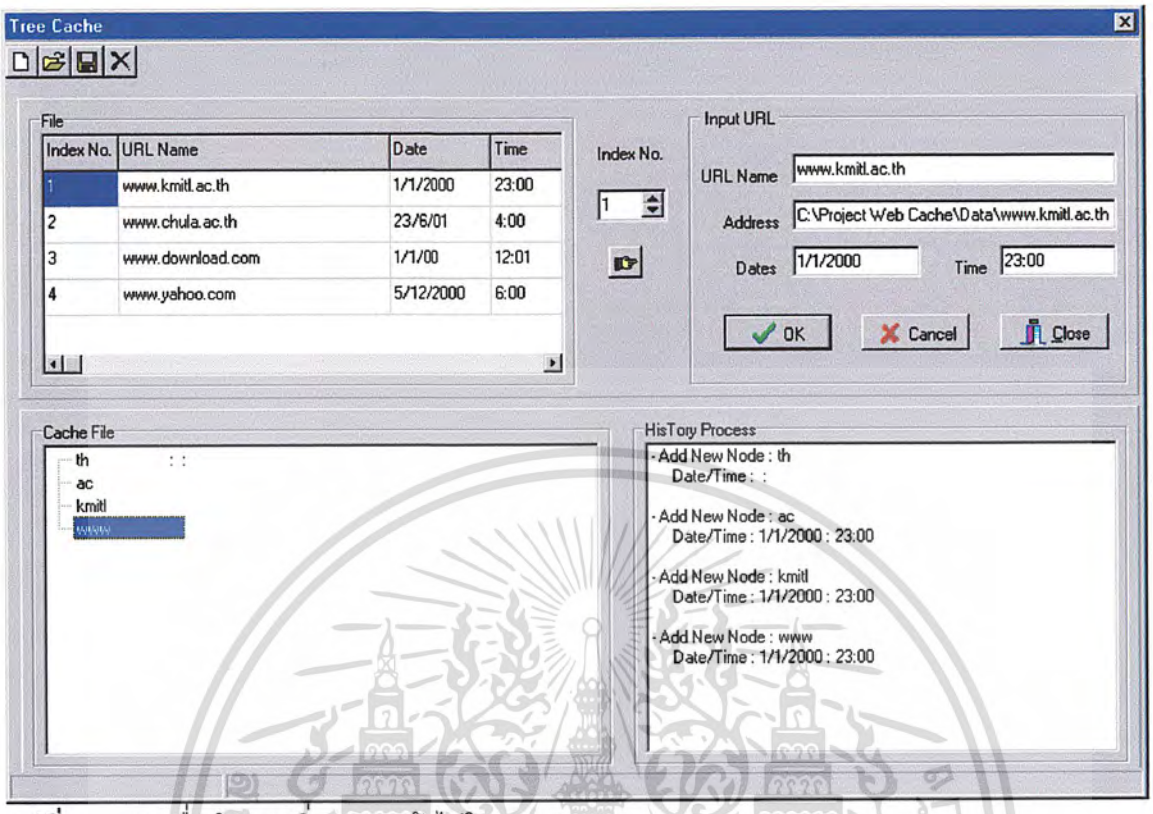
การรับ file ที่มาจากโปรแกรม Cache editor เพื่อนำมาเก็บใน Cache ซึ่งในโปรแกรมนี้จะแบ่งออกได้เป็น 4 ส่วน

1. file เป็น table ที่นำค่า file *.cac ที่ได้มาจากโปรแกรม Cache editor
2. input URL เป็นส่วนที่ให้ผู้ใช้สามารถเลือกได้ว่าต้องการที่จะนำ URL ใด ไปเก็บใน Cache ซึ่งจะมี index number เพื่อที่จะสามารถเลือก index นั้น เก็บลง Cache และมี URL size เพื่อที่จะกำหนด

   ขนาดของ Cacheว่า สามารถที่จะเก็บ URL ได้กี่ค่า

การ Insert URL ใน Project Tree

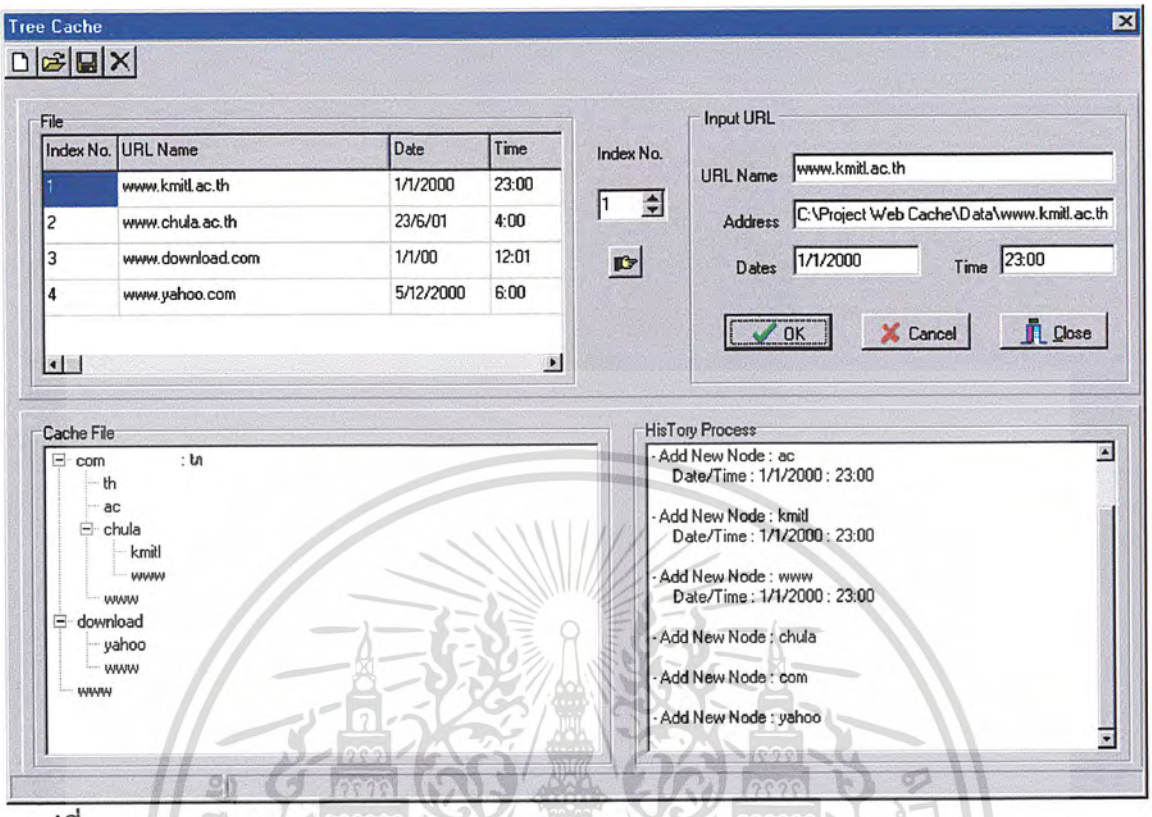
1. เมื่อทำการเพิ่ม www.kmitl.ac.th เข้าไปใน Cache ก็จะแสดงรูป Tree ตามหลักการที่เราได้ออกแบบไว้แล้ว



รูปที่ ก – 15 เมื่อทำการเพิ่ม URL เข้าไปใน Cache

การ UpDate URL ใน Project Tree

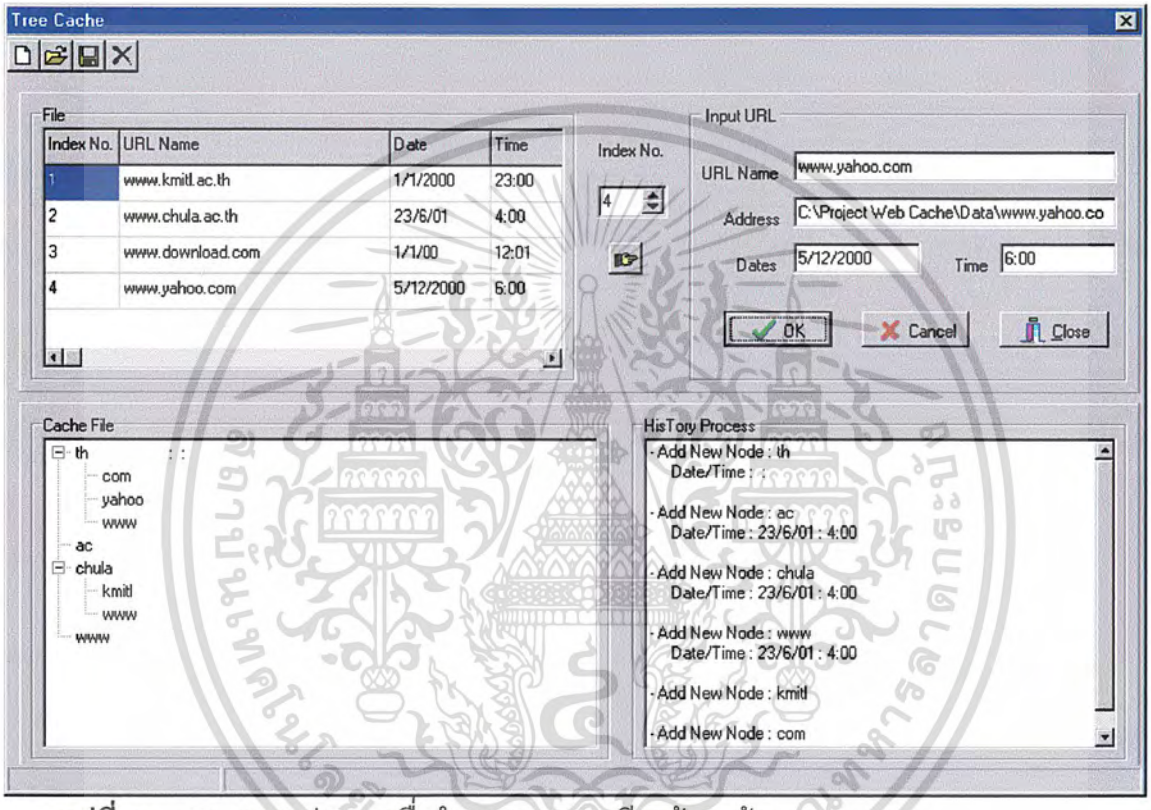
2. เมื่อมีการเรียกใช้ URL เดิมเข้าไปใน Cache ก็จะมีเปลี่ยนแปลงวันที่ของ URL นั้นและจะมีการย้าย Tree ไปอยู่ข้างขวาสุดของ Tree ตามแต่ละ Level ในที่นี้จะได้เพิ่ม www.kmitl.ac.th เข้าไปโดยที่ใน Cache ก็มี URL นี้อยู่แล้ว ก็จะทำการ UpDate วันเวลาของ URL นี้ตามรูป ก - 16



รูปที่ ก – 16 แสดงการ Update URL www.kmitl.ac.th

การ Delete URL ใน Project Tree

3. เมื่อ Cache เต็มจะทำการลบ URL ใน Tree ที่เก่าที่สุดทิ้งไป โดยดูจากวัน-เวลาของ URL นั้น ๆ แต่จากตามการออกแบบของเรานั้น ทำให้เราสามารถรู้ว่า URL ที่เก่าที่สุดนั้นเป็น URL ทางซ้ายสุดทุกครั้ง เมื่อ Cache เต็ม จึงจะทำการลบ URL www.dowmload.com และจะทำการเปรียบเทียบกันระหว่าง ค่าวัน-เวลาของ root ของแต่ล่ะ สาย



รูปที่ ก – 17 แสดงรูป Tree เมื่อทำการลบ URL เรียบร้อยแล้ว

บรรณานุกรม

นิรุธ อำนวยศิลป์.2542. คู่มือการเขียนโปรแกรม Delphi version 5.0 กรุงเทพฯ : ซัคเซส มีเดีย .

Adrain Low. Introduction Computer Vision And Image Processing. McGraw – Hill, 1991.

J. N. Kapur, P.K. Sahoo, and A.K.C. Wong. " A New Method for Gray – level picture Thresholding Using the Entropy of the Histogram, "Comp.Vision Graph.AndImage Proc.,Vol.29,1992.pp.273—285.

T.Pun." A New Method for Gray – level picture Thresholding Using the Entropy of the Histogram , "Signal Processing, Vol. 2,1980 . pp.223 – 237.

Wayne Niblack . An Intorduction Digital Image Processing. Prentice – Hall , 1986 .