บทที่ 2

แนวคิดและทฤษฎี

ในการที่จะออกแบบและพัฒนาโปรแกรมแสดงผลตัวอักษรไทยด้วยระบบเท็กซ์นั้นจำเป็นที่จะต้องมีความรู้พื้นฐานเกี่ยวกับองค์ประกอบที่จำเป็นในการสร้างโปรแกรมไทยเท็กซ์เสียก่อน

องค์ประกอบที่จำเป็นในการทำงานของโปรแกรมเท็กซ์โดยทั่วไปนั้นสามารถแบ่งออกได้เป็น 4 ส่วน ดังนี้

1. ส่วนที่ทำหน้าที่จัดการกับแฟ้มเอกสารให้อยู่ในรูปแบบที่ต้องการจะพิมพ์ ในส่วนนี้จะหมายถึงตัวโปรแกรมของเท็กซ์เองซึ่งจะทำหน้าที่เปลี่ยนแฟ้มเอกสารที่อยู่ในรูปแบบไวยากรณ์ที่เท็กซ์เข้าใจให้ไปอยู่ในรูปที่อุปกรณ์แสดงผลรู้จักได้

2. ส่วนของตัวอักษร (font) ที่ใช้ ซึ่งในส่วนนี้จะมีโปรแกรมเมตาฟอนต์เป็นตัวจัดการสร้างขึ้น การเรียกใช้ฟอนต์ของโปรแกรมเท็กซ์นั้น อาจจะเรียกใช้ฟอนต์ที่สร้างจากโพสท์คริปต์(postscript) หรือฟอนต์ประเภททรูไทพ์ (true type) ได้แต่ต้องมีโปรแกรมที่เปลี่ยนรูปแบบแฟ้มที่ได้จากการ ทำงานของเท็กซ์ให้เป็นในรูปแบบที่ฟอนต์ชนิดนั้นรู้จัก

3. ส่วนของซอฟท์แวร์อรรถประโยชน์ (utilities software) เป็นส่วนของโปรแกรมสนับสนุนในการทำงานของเท็กซ์และเมตาฟอนต์ เช่นการแปลงรูปแบบของแฟ้มจากประเภทหนึ่งไปเป็นอีกประเภทหนึ่ง โปรแกรมในส่วนนี้จะทำให้ผู้ใช้งานเท็กซ์มีความสะดวกในการที่จะนำแฟ้มเอกสารที่จัดทำด้วยเท็กซ์หรือเมตาฟอนต์ที่ทำงานอยู่บนระบบปฏิบัติการหนึ่งสามารถเปลี่ยนไปใช้งานกับระบบปฏิบัติการอื่นๆ ได้

4. ส่วนของซอฟท์แวร์ที่ควบคุมการแสดงผลออกทางอุปกรณ์ต่างๆ (output device driver software) เป็นส่วนของโปรแกรมที่ดำเนินการนำแฟ้มที่ได้จากกระบวนการของเท็กซ์ไปแสดงผลลงบนอุปกรณ์ต่างๆ ในส่วนนี้เป็นเรื่องของอุปกรณ์แสดงผลแต่ละชนิดที่ต้องจัดหาซอฟท์แวร์ไดรว์เวอร์ซึ่งขึ้นอยู่กับฮาร์ดแวร์แต่ละชนิด

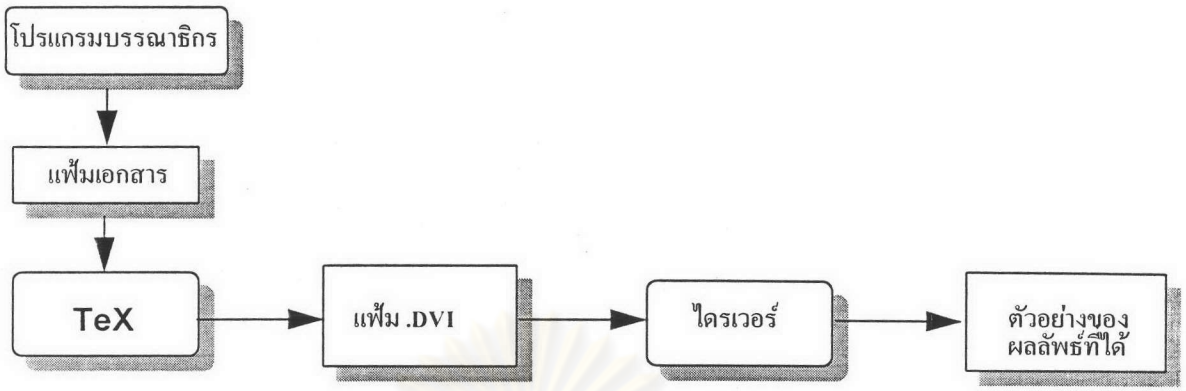
ด้วยองค์ประกอบข้างต้นซึ่งถือเป็นหัวใจในการพัฒนาระบบเท็กซ์นี้แสดงให้เห็นว่า ระบบเท็กซ์จะมีความยืดหยุ่นกับการใช้และพัฒนาค่อนข้างมาก เช่นสมมุติว่ามีอุปกรณ์แสดงผลแบบใหม่เกิดขึ้นและต้องการที่จะแสดงผลออกทางอุปกรณ์นั้นก็เพียงแต่สร้างดีไวซ์ไดรว์เวอร์สำหรับอุปกรณ์ชนิดนั้นขึ้นมาเท่านั้น หรือถ้าต้องการฟอนต์ชนิดใหม่ก็อาจจะใช้เมตาฟอนต์เป็นตัวสร้างฟอนต์ขึ้นมาตามความพอใจได้ หรือถ้าไม่ต้องการที่จะสร้างฟอนต์เองแต่ต้องการที่จะใช้ฟอนต์ประเภททรูไทพ์ชนิดต่างๆ ก็เพียงแต่หาซอฟท์แวร์ที่สามารถแปลงรูปแบบของแฟ้มที่เกิดขึ้นในระบบเท็กซ์ให้เป็นรูปแบบที่เหมาะสมกับการใช้งานของฟอนต์ประเภทนั้นได้เช่นกัน

การพัฒนาโปรแกรมแสดงผลภาษาไทยบนระบบเท็กซ์นี้ก็อาศัยองค์ประกอบที่กล่าวข้างต้นนี้เช่นกัน โดยจะเข้าไปข้องเกี่ยวเฉพาะการพัฒนาตัวของเท็กซ์และการพัฒนาฟอนต์เท่านั้น ซึ่งวิธีการพัฒนาจะนำเสนอในบทต่อไป อย่างไรก็ตามการพัฒนาโปรแกรมเพื่อที่จะจัดเรียงพิมพ์ด้วยระบบเท็กซ์หรือการประมวลผลคำด้วยโปรแกรมประมวลผลคำใดๆก็ตาม สิ่งที่ผู้ใช้ต้องการนอกเหนือไปจากความสวยงามของชิ้นงานแล้ว ความสละสลวยของภาษาที่ได้ในงานพิมพ์นั้นก็เป็นสิ่งที่ผู้ใช้ไม่สามารถมองข้ามได้เช่นกัน ในตัวของเท็กซ์เองจะมีอัลกอริธึมที่จัดการกับการแบ่งคำที่เป็นภาษาอังกฤษด้วยไฮเฟนโดยเฉพาะ ดังนั้นถ้าต้องการที่จะพัฒนาโปรแกรมแสดงผลตัวอักษรภาษาไทยซึ่งต่อไปจะเรียกแทนว่าโปรแกรมไทยเท็กซ์จะไม่สามารถหลีกเลี่ยงในเรื่องของการแบ่งคำหรือจัดคำในแต่ละบรรทัด ได้เช่นกัน จึงจำเป็นที่จะต้องศึกษาแนวความคิดและทฤษฎีของการตัดคำนี้รวมอยู่ด้วย

จากที่กล่าวมาสามารถแยกหัวข้อของการศึกษาเพื่อที่จะพัฒนาโปรแกรมไทยเท็กซ์ได้เป็น 3 เรื่องใหญ่ คือ การศึกษาถึงการทำงานของเท็กซ์ การทำงานของเมตาฟอนต์ และ การตัดคำ

1. <u>การทำงานของ TeX</u>

หน้าที่ของระบบเท็กซ์ก็คือการเรียกใช้งานของเครื่องมือที่มีอยู่ในระบบของเท็กซ์เอง เช่น ไฟล์ชนิดต่างๆ ดีไวซ์ไดร์เวอร์ต่างๆ นำมาผลิตเป็นเอกสารที่มีรูปแบบซับซ้อนตามที่ต้องการ ดังนั้นการที่เราจะสร้างเอกสารให้มีรูปแบบตามที่ต้องการได้จำเป็นจะต้องรู้จักกับส่วนประกอบหลักของเท็กซ์ซึ่งแสดงไว้ดังรูปที่ 2.1



รูปที่ 2.1 แสดงการทำงานโดยทั่วไปของโปรแกรมเท็กซ์

1.1 *การสร้างแฟ้มเอกสาร(Document File)*

เป็นขั้นตอนเริ่มแรกของการทำงานในการสร้างรูปแบบการพิมพ์ของระบบเท็กซ์ ซึ่งเราสามารถเลือกโปรแกรมบรรณาธิกรที่จะใช้สร้างได้แทบจะทุกประเภทตามแต่ที่จะมีในระบบปฏิบัติการของเครื่องคอมพิวเตอร์ชนิดนั้นๆ อย่างไรก็ตามควรที่จะเลือกโปรแกรมบรรณาธิกรที่สร้างแฟ้มในลักษณะของเท็กท์ไฟล์ กล่าวคือเป็นแฟ้มข้อมูลที่เก็บข้อมูลแบบสตริงของอักขระ (Character String) ล้วนเพื่อที่จะไม่ต้องมีปัญหาของส่วนหัวของไฟล์ (File Header) ซึ่งระบบเท็กซ์อาจจะไม่รู้จัก

1.2 *การประมวลผลด้วยเท็กซ์*

หลังจากที่ได้จัดเตรียมแฟ้มเอกสารเรียบร้อยแล้วก็ถึงขั้นตอนของการเรียก โปรแกรมเท็กซ์เพื่อที่จะทำการประมวลผลแฟ้มเอกสารที่ได้จัดเตรียมไว้ให้เปลี่ยนสภาพเป็นแฟ้มที่ไม่ขึ้นกับอุปกรณ์ชนิดใดๆ (Device Independent File) ซึ่งหลังจากผ่านขั้นตอนนี้แล้วจะได้แฟ้มที่มีนามสกุล .dvi ในขั้นตอนดังกล่าวนี้อาจจะมีการเกิดความผิดพลาดขึ้นได้เนื่องจากการสะกดคำสั่งผิด การผิดไวยากรณ์ของเท็กซ์ หรืออาจจะเกิดจากความไม่เข้าใจโครงสร้างของชุดคำสั่งของเท็กซ์ดีพอก็ได้ ข้อผิดพลาดดังกล่าวนี้จะแสดงออกมาทางจอภาพและจะมีการบันทึกลงได้ในแฟ้มที่มีนามสกุล.log เพื่อให้ผู้ใช้สามารถดูและทำการแก้ไขแฟ้มเอกสารให้ถูกต้องต่อไป

โปรแกรมเท็กซ์ที่ใช้ประมวลผลนั้นมีมากมายหลายรุ่นขึ้นอยู่กับระบบคอมพิวเตอร์ที่ใช้เช่น TeX และ LaTeX สำหรับใช้บนระบบปฏิบัติการยูนิกซ์, emTeX สำหรับระบบปฏิบัติการแบบดอส เป็นต้น นอกจากนี้ยังได้มีการพัฒนาออกไปเพื่อให้ใช้ได้กับวินโดวส์และแม็คอินทอชได้ด้วย แต่ที่ถือเป็นต้นแบบของระบบนี้ก็คือ TeX สำหรับโปรแกรม LaTeX นั้นเกิดจากการพัฒนาชุดคำสั่งของ TeX ให้ผู้ใช้สามารถใช้งานได้ง่ายและสะดวกยิ่งขึ้น ดังนั้นรูปแบบของคำสั่งบางคำสั่งจึงแตกต่างกันไปบ้าง

1.3 *การใช้คำสั่งมาโคร (Macro Instruction)*

คำสั่งควบคุมลำดับการทำงานต่างๆที่เราพิมพ์แทรกเข้าไปในแฟ้มเอกสารนี้ จะถูกกำหนดโดยชุดของมาโคร (Macro Package) ชุดของมาโครนี้ประกอบด้วยคำสั่งต่างๆที่ใช้ในเท็กซ์ซึ่งอาจจะถูกพัฒนาขึ้นโดยภาษาระดับสูง(ภาษาซี) หรืออาจจะถูกพัฒนาขึ้นจากชุดคำสั่งง่ายๆ มาเรียงต่อกัน เพื่อให้สามารถทำงานอย่างใดอย่างหนึ่งได้สะดวกยิ่งขึ้น โดยปกติชุดของมาโครจะถูกเก็บอยู่ในแฟ้มรูปแบบ (Format File) เมื่อเวลาโปรแกรมเท็กซ์เริ่มทำงานจะทำการแปลชุดคำสั่งเหล่านี้แล้วเก็บไว้ในแฟ้มรูปแบบเพื่อที่จะได้ใช้ตอนทำการประมวลผลด้วยเท็กซ์ต่อไป คำสั่งที่ถูกพัฒนาขึ้นโดยภาษาซีที่ติดมากับโปรแกรมเท็กซ์นั้นเรียกว่าคำสั่งดั้งเดิม (Primitive Instruction) คำสั่งประเภทนี้จะไม่สามารถแก้ไขเป็นอย่างอื่นได้นอกจากจะทำการพัฒนาระบบเท็กซ์ขึ้นมาใหม่ทั้งระบบ คำสั่งประเภทนี้ได้แก่ คำสั่งที่จัดการเกี่ยวกับรูปแบบของอินพุตและเอาท์พุตของตัวอักษร เป็นต้น ส่วนคำสั่งอีกประเภทหนึ่งนั้นจะถูกพัฒนาขึ้นมาจากคำสั่งดั้งเดิมเรียกว่าคำสั่งทั่วไป (Plain Instruction) ในโปรแกรม TeX จะเก็บคำสั่งเหล่านี้ไว้ในแฟ้มข้อมูลที่ชื่อว่า plain.tex และในโปรแกรม LaTeX จะเก็บไว้ในแฟ้ม latex.tex คำสั่งทั่วไปนี้มีข้อดีคือรูปแบบการใช้งานจะง่ายกว่าแบบคำสั่งดั้งเดิมแต่มีข้อเสียอยู่ตรงที่จะสิ้นเปลืองเวลาในการประมวลผลมากกว่าเนื่องจากจะต้องทำการขยายคำสั่งที่ซ้อนอยู่ภายในออกมาให้อยู่ในรูปของคำสั่งดั้งเดิมหลายๆคำสั่งก่อนจึงจะทำการประมวลผลได้ อย่างไรก็ตามถ้าผู้ใช้งานมีความรู้ในเรื่องคำสั่งมาโครดีเพียงพอก็อาจจะเขียนชุดคำสั่งนี้ให้ทำงานตามที่ต้องการได้เช่นการสร้างโปรแกรม LaTeX นั้นจะอาศัยการเขียนคำสั่งมาโครที่โปรแกรม TeX มีมาให้แต่เพียงอย่างเดียวโดยไม่เข้าไปยุ่งเกี่ยวกับโครงสร้างหรือระบบภายในของตัวโปรแกรม TeX แม้แต่น้อย รายละเอียดของโปรแกรมดังกล่าวสามารถศึกษาได้จากแฟ้ม latex.tex

นอกจากนี้ การพัฒนาโปรแกรมไทยเท็กซ์นั้น ส่วนหนึ่งของการพัฒนาก็จำเป็นต้องอาศัยการเขียนชุดคำสั่งมาโครด้วยเช่นเดียวกัน แต่การที่ไม่สามารถพัฒนาโปรแกรมไทยเท็กซ์ทั้งหมดด้วยการเขียนชุดคำสั่งมา

โครนั้นก็เนื่องจากในส่วนของการพัฒนาอัลกอริทึมสำหรับตัดคำภาษาไทยนั้นจำเป็นต้องใช้ภาษาที่มีความคล่องตัวมากกว่าชุดคำสั่งมาโครของเท็กซ์ รายละเอียดดังกล่าวจะได้นำเสนอในบทต่อไป

1.4 *การเรียกใช้ตัวอักษรแบบต่างๆ*

จุดเด่นที่น่าสนใจของระบบเท็กซ์ก็คือลักษณะของตัวอักษรที่เรียกใช้ โดยปกติแล้วในซอฟท์แวร์ของโปรแกรมเท็กซ์เองจะมีชุดของตัวอักษรหรือฟอนต์ให้เลือกใช้มากมายหลายขนาดตามต้องการ แต่ก็ยังมีซอฟท์แวร์ให้ผู้ใช้สามารถสร้างฟอนต์ขึ้นมาใช้เองได้อีกด้วย ฟอนต์ที่จะถูกเรียกใช้งานโดยโปรแกรมเท็กซ์จะต้องอยู่ในรูปของเมตริกฟอนต์ (TeX Font Matrix) หรือแฟ้มที่มีนามสกุลเป็น .tfm ซึ่งในแฟ้มนี้จะเก็บรายละเอียดของฟอนต์แต่ละตัวที่ใช้เช่นความกว้าง ความสูง รูปร่าง เป็นต้น ซอฟท์แวร์ที่ใช้สำหรับสร้างฟอนต์นี้คือ METAFONT ซึ่งจะได้กล่าวถึงต่อไป

ตามปกติแล้วถ้าไม่ใช้คำสั่งอื่น ในชุดคำสั่งมาโครจะเลือกใช้ฟอนต์ชนิดคอมพิวเตอร์โมเดิร์นฟอนต์ (Computer Modern fonts)เสมอจนทำให้เข้าใจว่าเท็กซ์จะใช้ได้แต่เพียงฟอนต์ชนิดนี้เพียงอย่างเดียวซึ่งเป็นสิ่งที่เข้าใจผิดเป็นอย่างมากการจะใช้ฟอนต์ชนิดอื่นนั้นจะกระทำได้โดยไม่ยากนักแต่บางครั้งก็ทำให้เกิดความสับสนบ้างและก็อาจจะทำให้งานพิมพ์ที่ได้ผิดไปจากความต้องการของผู้ใช้ ขั้นตอนการทำงานของเท็กซ์ในการเรียกใช้ฟอนต์ชนิดต่างๆมีดังนี้

1.4.1 กำหนดชนิดของฟอนต์ด้วยคำสั่งมาโคร

ขั้นตอนนี้จะเป็นการบอกให้คำสั่งมาโครของเท็กซ์ทราบว่าชนิดของฟอนต์ที่จะใช้นั้นเป็นชนิดใดเช่น สมมุติว่ามีฟอนต์ตัวอักษรภาษาไทยอยู่ชุดหนึ่งซึ่งเก็บอยู่ในแฟ้มที่มีชื่อว่า san.tfm ถ้าต้องการจะเรียกใช้ฟอนต์ชุดนี้ในแฟ้มเอกสารของก็สามารถทำได้โดยสร้างคำสั่งมาโครขึ้นให้มีชื่อว่า thai จากนั้นก็กำหนดค่าให้มาโครที่มีชื่อว่า thai นี้เป็นชื่อของฟอนต์ san.tfm ดังนี้

```
\font\thai=san
```

หลังจากนั้นไม่ว่าเมื่อใดที่เราต้องการใช้ฟอนต์ชนิดนี้ก็เพียงแต่เรียกใช้คำสั่งมาโคร เท่านั้น ฟอนต์ที่ตามหลังคำสั่งดังกล่าวก็จะเป็นฟอนต์ชนิด san ทั้งหมดเว้นแต่จะมีการกำหนดฟอนต์ชนิดอื่นต่อไปอีก

1.4.2 เท็กซ์จะทำการโหลดรายละเอียดของฟอนต์ชนิดที่กำหนดไว้ในแฟ้ม san.tfm

ขั้นตอนนี้เท็กซ์จะทำการค้นหาแฟ้มดังกล่าวในไดเร็คตอรี่ที่กำหนดไว้ ตอนเริ่มติดตั้งระบบ สิ่งที่ต้องระลึกไว้เสมอก็คือ เท็กซ์จะต้องการเฉพาะแฟ้มที่มีนามสกุล .tfm เท่านั้น ถ้าไม่เจอแฟ้มดังกล่าว เท็กซ์จะแสดงข้อผิดพลาดและหยุดการทำงาน

1.4.3 เท็กซ์จะทำการสร้างแฟ้ม .dvi ขึ้น

ในแฟ้ม .dvi จะเก็บรายละเอียดเกี่ยวกับชนิด อัตราขยายของฟอนต์ที่ใช้

1.4.4 ไดรว์เวอร์จะทำหน้าที่จัดการกับแฟ้ม DVI

โดยจะกำหนดแฟ้มของฟอนต์ (font file) สำหรับฟอนต์แต่ละตัวตามขนาดของฟอนต์ที่เรียกใช้ ไดรว์เวอร์บางตัวอาจจะตรวจสอบกับรายชื่อของฟอนต์ที่มีอยู่ในตัวของเครื่องพิมพ์แต่ละชนิดด้วย ก่อนที่จะทำการกำหนดแฟ้มของฟอนต์ นอกจากนี้ไดรว์เวอร์บางตัวยังมีความสามารถที่จะเลือกฟอนต์ทดแทนได้ในกรณีที่ตรวจสอบแล้วไม่พบฟอนต์ที่อ้างถึงแฟ้มที่ไดรว์เวอร์ต้องการนั้นจะมีนามสกุลแตกต่างกันไปเช่น.pk, .pxl, .gf เป็นต้น ทั้งนี้ขึ้นอยู่กับชนิดของไดรว์เวอร์นั้นๆ สำหรับในการวิจัยครั้งนี้ใช้โปรแกรม emTeX เป็นตัวทดสอบ และไดรว์เวอร์ที่ติดมากับซอฟท์แวร์ชุดนี้จะต้องการใช้แฟ้ม .pk

1.4.5 ไดรว์เวอร์จะทำหน้าที่ผลิตเอาท์พุตออกมาให้เหมาะสมกับอุปกรณ์เอาท์พุตประเภทนั้นๆ

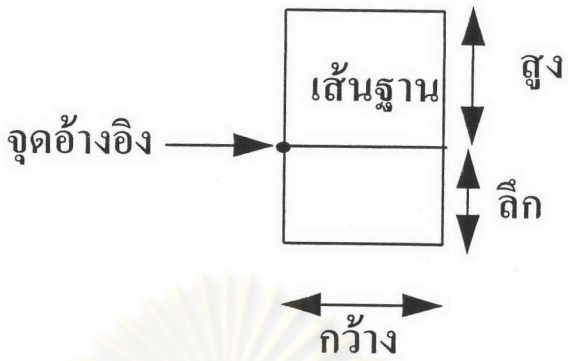
ข้อที่น่าจดจำอีกข้อหนึ่งก็คือเท็กซ์จะไม่ใช่ตัวที่ทำหน้าที่ผลิตเอาท์พุตโดยตรง เท็กซ์จะทำหน้าที่แต่เพียงสร้างแฟ้ม .dvi เท่านั้น และแฟ้ม .dvi นี้ก็ไม่สามารถแสดงผลออกมาเป็นเอาท์พุตได้เลยในทันที จะต้องนำไปผ่านไดรว์เวอร์เสียก่อนจึงจะออกไปเป็นเอาท์พุตได้ การแสดงผลของไดรว์เวอร์จะไม่เกี่ยวข้องกันกับตัวโปรแกรมเท็กซ์แต่จะเกี่ยวข้องกับฮาร์ดแวร์ของอุปกรณ์นั้นๆ ดังนั้นถ้าเรามีแฟ้มเอกสารที่จัดเตรียมไว้ด้วยโปรแกรมเท็กซ์เราสามารถนำไปประมวลผลด้วยระบบเท็กซ์ที่มีลักษณะของไวยากรณ์หรือโครงสร้างของคำสั่งใกล้เคียงกัน หรือเหมือนกันได้ทุกระบบไม่ว่าจะเป็นระบบปฏิบัติการแบบใดก็ตาม และจะให้ผลออกมาเป็นแฟ้ม .dvi เช่นเดียวกัน

2. การทำงานของเมตาฟอนต์

เนื่องจากการออกแบบการทำงานของเท็กซ์จะใช้หลักการของกล่องและกาว กล่าวคือเท็กซ์จะไม่สนใจว่ามีอะไรอยู่ในกล่องบ้าง แต่จะสนใจว่าจะนำกล่องแต่ละกล่องมาประกอบรวมกันอย่างไรเท่านั้น ดังนั้น ถ้าเท็กซ์ทำการอ้างอิงถึงกล่องดังกล่าวแต่ละกล่องได้ก็สามารถดำเนินการนำมาวางประกอบกันได้เสมอ

กล่องแต่ละกล่องอาจจะประกอบไปด้วยตัวอักษร(letter)หรือสัญญลักษณ์ (symbol) ....คำ (word) ประโยค (sentence) หรือย่อหน้า (paragraph) ก็ได้ สำหรับหน่วยเล็กที่สุดที่เท็กซ์ถือว่าเป็นกล่องก็คือตัวอักษรแต่ละตัวหรือสัญญลักษณ์

คำว่ากล่อง แท้จริงแล้วเป็นเพียงกรอบที่มีเพียง 2 มิติที่มองเห็นได้ในระนาบเดียวเท่านั้น แต่ในความหมายของเท็กซ์นี้จะมี3มิติคือความกว้าง ความสูง และความลึกโดยมีการกำหนดจุดอ้างอิงหนึ่งจุด เส้นที่ลากผ่านจุดนี้ในแนวระดับจะเรียกว่าเส้นฐาน มิติความกว้างก็คือระยะทางที่ห่างจากจุดอ้างอิงไปทางขวา มิติความยาวคือ ระยะทางเหนือเส้นฐาน สำหรับความลึกนั้นเท็กซ์กำหนดให้คือระยะทางใต้เส้นฐาน องค์ประกอบต่างๆของกล่องจะเป็นดังรูปที่ 2.2 ในกล่องนี้จะบรรจุตัวอักษรหรือสัญญลักษณ์ที่สร้างขึ้น ตัวอักษรหรือสัญญลักษณ์ดังกล่าวอาจจะอยู่ในกล่องที่สร้างขึ้นทั้งหมดหรืออาจจะมีบางส่วนที่เกินออกมานอกกล่องบ้างก็ได้สำหรับตัวอักษรหนึ่งชุดนั้นไม่มีข้อกำหนดว่ามิติของแต่ละกล่องจำเป็นต้องเท่ากัน มิติของกล่องแต่ละกล่องนั้นจะขึ้นกับรูปร่างลักษณะของตัวอักษรแต่ละตัวตลอดจนความสวยงามของตัวอักษรแต่ละชุดเป็นเกณฑ์สำคัญ การอ้างอิงถึงกล่องแต่ละกล่องจะใช้รหัสแอสกี้ สิ่งที่ทำให้เท็กซ์สามารถเรียกใช้กล่องเหล่านี้ได้ก็คือการอ้างถึงรหัสแอสกี้นั่นเอง ดังนั้นเท็กซ์จะไม่มีโอกาสรู้ได้ว่าสิ่งที่อยู่ในกล่องจะมีรูปร่างหน้าตาเป็นอย่างไร เพียงแต่รู้จักอ้างอิงถึงรหัสสำหรับตัวอักษรหรือสัญญลักษณ์ก็เพียงพอแล้ว

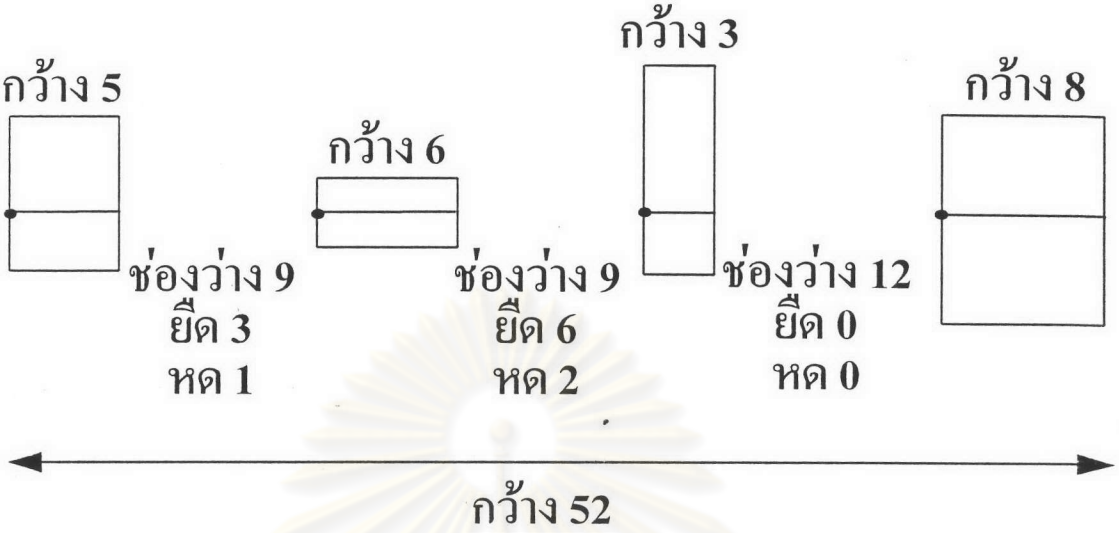


รูปที่ 2.2 แสดงภาพองค์ประกอบของกล่อง

สิ่งที่สำคัญสำหรับเท็กซ์ก็คือการนำกล่องเหล่านี้มาเรียงต่อกันเพื่อให้เป็นรูปแบบของงานพิมพ์ตามความต้องการของผู้ใช้ การจัดเรียงกล่องที่ต้องการนี้ก็อาจจะเปรียบเทียบได้กับการจัด เรียงพิมพ์หรือแสดงรูปภาพหรือข้อความบนบอร์ดตามงานนิทรรศการต่างๆกล่าวคือเริ่มต้นเราจะนำรูปภาพเหล่านั้นมากำหนดเสียก่อนว่าจะวางลงในตำแหน่งใด หลังจากที่ได้ตำแหน่งตามที่ต้องการ ทากาว แล้วจึงนำภาพหรือข้อความนั้นแปะทับลงบนกระดานว่างนั้น บางครั้งในหนึ่งรูปภาพที่นำมาแปะอาจจะมีข้อความหรือรูปภาพย่อยๆ ติดกาวอยู่บนรูปภาพนั้นก่อนแล้วก็เป็นได้ หลักการทำงานของเท็กซ์ก็เป็นเช่นเดียวกัน การกำหนดตำแหน่งที่สัมพันธ์กันของแต่ละกล่องนั้นจะอ้างอิงจากเส้นฐานเป็นเกณฑ์ ความแตกต่างอย่างหนึ่งของการติดรูปภาพบนบอร์ดและการเรียงพิมพ์ของเท็กซ์นั้นก็คือ เท็กซ์จะมีระยะยืดหรือระยะหดเพื่อทำให้ช่องไฟระหว่างตัวอักษรแต่ละตัวเป็นไปอย่างสวยงามตามขนาดตัวอักษรที่ใช้ในแต่ละย่อหน้า ในขณะที่รูปภาพที่ติดอยู่บนบอร์ดจะถูกกำหนดตายตัวไม่สามารถเว้นระยะได้อีก

เพื่อที่จะทำความเข้าใจของหลักการของกาวในระบบเท็กซ์ได้ดียิ่งขึ้นจะพิจารณาได้ จากตัวอย่างต่อไปนี้

สมมุติว่ามีกล่องอยู่ 4 กล่องและมีช่องว่างสำหรับติดกาวอยู่ 3 ช่องดังรูปที่ 2.3



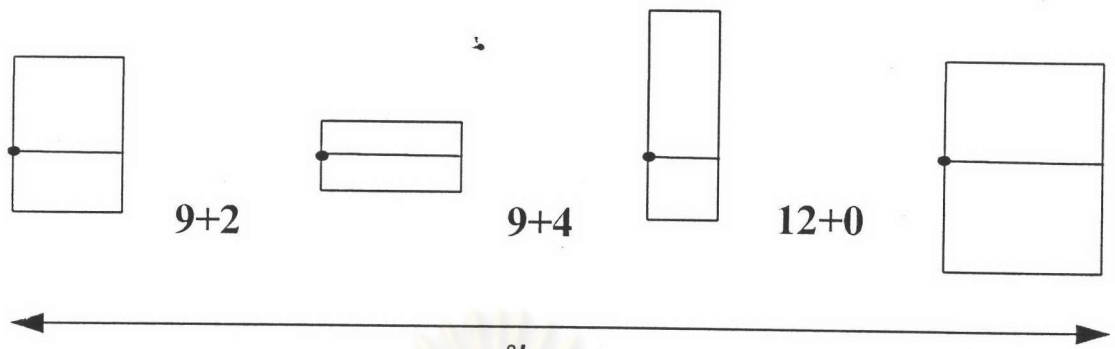
รูปที่ 2.3 แสดงภาพตัวอย่างของกล่องและกาว

ช่องแรกของกาวจะมีขนาดเปรียบเทียบเป็นช่องว่างได้ 9 หน่วยและมีระยะยืด 3 หน่วย ระยะหด 1 หน่วย ช่องที่สองจะมีขนาด 9 หน่วย ระยะยืด 6 หน่วย ระยะหด 2 หน่วย ช่องสุดท้าย จะมีขนาด 12 หน่วย ระยะยืด 0 หน่วย ระยะหด 0 หน่วยซึ่งหมายความว่าไม่สามารถยืดหรือหดได้

ความกว้างของกล่องและกาวในภาพนี้พิจารณาเฉพาะช่องว่างรวมทั้งหมดคือ

5+9+6++9+3+12+8 = 52 หน่วย

ซึ่งจะเรียกว่าความกว้างธรรมชาติ (natural width) ในแนวราบ อย่างไรก็ตามเท็กซ์ก็สามารถที่จะทำให้ความกว้างรวมทั้งหมดเป็น 58 หน่วยได้ถ้ามีการนำส่วนยืดที่กำหนดไว้มาบวกเพิ่มเข้าไปอีก 6 หน่วย จากตัวอย่างข้างต้นส่วนยืดที่กำหนดไว้มีขนาด 3+6+0 = 9 หน่วย ดังนั้นเพื่อที่จะให้ผลรวมของความกว้างของทั้งบรรทัดเพิ่มขึ้นอีก 6 หน่วยเท็กซ์จะทำการคูณส่วนยืดของกาวแต่ละช่องด้วย 6/9 ความกว้างของกาวช่องแรกจะเป็น 9+(6/9)x3= 11 หน่วย ความกว้างของช่องที่สองจะเป็น 9+(6/9)x6= 13 หน่วย ส่วนในช่องสุดท้ายเนื่องจากส่วนยืดมีค่าเป็นศูนย์ ค่าของความกว้างของกาวจึงเป็น 12 หน่วยเท่าเดิม ผลของการยืดขนาดของกาวที่ได้จะแสดงได้ดังรูปที่ 2.4



รูปที่ 2.4 แสดงการขยายระยะยืดของกาว

ในขณะเดียวกัน ถ้าต้องการหดความกว้างของกาวแต่ละช่องให้ความกว้างของแนวระดับทั้งบรรทัดรวมกันแล้วเท่ากับ 51 หน่วย จะต้องทำการคูณส่วนหดของกาวด้วย 1/3 สำหรับช่องแรกและ 2/3 สำหรับช่องที่สอง ส่วนในช่องสุดท้ายจะไม่สามารถหดลงได้เนื่องจากส่วนหดมีค่า เป็นศูนย์

ความกว้างของกาวในแต่ละช่องนั้นจะไม่สามารถหดเล็กลงเกินกว่าค่าส่วนหดในแต่ละช่องนั้น ในทางกลับกัน ความกว้างของกาวในแต่ละช่องสามารถยืดได้เกินกว่าค่าของส่วนยืดเมื่อค่า ของส่วนยืดนั้นยังคงมีค่าเป็นบวก

จากที่กล่าวมาข้างต้นแล้วจะเป็นหลักการทำงานของเท็กซ์ จะเห็นว่าตัวเท็กซ์เองจะดำเนินการประมวลผลกับสิ่งที่อยู่นอกกล่อง เช่นการนำกล่องมาเรียงต่อกันในแนวราบหรือแนวดิ่ง การยืดหรือหดระยะระหว่างกล่อง แต่เท็กซ์จะไม่สามารถเข้าไปจัดการกับสิ่งที่อยู่ในกล่องได้ สิ่งที่จะทำหน้าที่จัดการกับสิ่งที่อยู่ในกล่องคือเมตาฟอนต์เท่านั้น

การสร้างฟอนต์นั้นจะมีวิธีหลักอยู่ 2 วิธีคือการสร้างฟอนต์แบบบิตแมพและการสร้างฟอนต์โดยใช้เวคเตอร์ แต่ละวิธีจะมีทั้งข้อดีและข้อเสียอยู่ในตัว วิธีบิตแมพจะดีตรงที่ผู้ออกแบบสามารถมองเห็นภาพของฟอนต์ที่สร้างขึ้นได้ชัดเจนและกำหนดรายละเอียดต่างๆได้ง่าย แต่จะมีข้อเสียคือเมื่อทำการย่อหรือขยายขนาดของฟอนต์แล้ว คุณภาพของฟอนต์ที่ได้จะไม่เหมือนกับขนาดตอนเริ่มต้นออกแบบ สำหรับฟอนต์แบบเวคเตอร์นั้นจะมีข้อดีคือเมื่อทำการย่อหรือขยายแล้วจะให้คุณภาพเหมือนหรือใกล้เคียงกับขนาดที่เริ่มต้นออกแบบมากกว่า ข้อเสียของฟอนต์แบบเวคเตอร์ก็คือการออกแบบทำได้ยาก ผู้ออกแบบต้องมีความรู้เกี่ยวกับการ

ใช้เวคเตอร์และระบบแกนอ้างอิง ตลอดจนคำสั่งที่จะใช้มากพอสมควร สำหรับวิธีการของเมตาฟอนต์จะเป็นวิธีการของเวคเตอร์ ผู้ออกแบบจะต้องเข้าใจขั้นตอนการออกแบบบ้างพอสมควรดังนี้

2.1 *ระบบแกนอ้างอิงและพิกัด*

ในการที่จะบอกคอมพิวเตอร์ว่าต้องการจะลากเส้นอย่างไรนั้น จำเป็นที่จะต้องมีวิธีการที่จะอธิบายว่าจุดที่จะลากเส้นผ่านนั้นอยู่ตรงไหนบ้าง เมตาฟอนต์จะใช้วิธีการอ้างอิงด้วยพิกัดคาร์ทีเซียน (Cartesian coordinates) ตำแหน่งของจุดจะถูกกำหนดโดยแกน X ซึ่งหมายถึงจำนวนหน่วยที่วิ่งไปทางขวามือจากจุดอ้างอิง และแกน Y ซึ่งหมายถึงจำนวนหน่วยที่วิ่งขึ้นข้างบนเหนือจุดอ้างอิง ในวิธีการสร้างภาพของเมตาฟอนต์นั้นสิ่งที่จำเป็นต้องกระทำคือการวาดรูปร่างของภาพที่ต้องการลงบนกระดาษกราฟอย่างคร่าวๆ จากนั้นก็ทำการกำหนดจุดที่สำคัญลงบนภาพที่วาดนั้นเพื่อความสะดวกก็อาจกำหนดให้เป็นตัวเลขจำนวนนับ หลังจากนั้นก็จะเป็นการเขียนโปรแกรมด้วยคำสั่งของเมตาฟอนต์เพื่อที่จะบรรยายว่า จุดที่กำหนดนั้นอยู่ที่ตำแหน่งใดในระบบแกนอ้างอิง และกำหนดว่าเส้นตรง หรือเส้นโค้งอะไรบ้างที่ถูกกำหนดให้ผ่านจุดเหล่านั้น

ภายในตัวเมตาฟอนต์เองจะเสมือนมีกระดาษกราฟอยู่ภายในกรอบสี่เหลี่ยมเล็กๆของกระดาษจะเรียกว่าพิกเซล (pixel) ผลลัพธ์ที่เมตาฟอนต์ให้ออกมาจะอยู่ในรูปของสถานะที่เป็นขาวหรือดำของพิกเซลเหล่านี้ เนื่องจากการใช้ระบบแกนอ้างอิงจะต้องมีหน่วยสำหรับบอกความยาว หรือระยะทางจากจุดอ้างอิงจุดหนึ่งไปยังจุดใดๆ เพื่อความสะดวกเมตาฟอนต์จะใช้ 1 หน่วยของพิกเซลเหล่านี้แทนระยะทางหนึ่งหน่วยที่อ้างถึง

การกำหนดระยะทางบนระบบแกนอ้างอิงนี้ไม่จำเป็นที่จะต้องกำหนดระยะทางเป็นเลขจำนวนเต็มเสมอไป อาจจะกำหนดเป็นจุดทศนิยมก็ได้ คอมพิวเตอร์จะสามารถทำงานได้ในระดับความละเอียดของจำนวนเท่าของ 1/65536 หรือประมาณ 0.00002 ของความกว้างของพิกเซล แต่เมตาฟอนต์จะไม่สามารถแบ่งให้ในหนึ่งพิกเซลมีได้ทั้งขาวและดำในขณะเดียวกัน สิ่งที่สามารถกำหนดได้ก็คือจะต้องเป็นดำทั้งหมดหรือขาวทั้งหมดเท่านั้น

ความละเอียดของจำนวนพิกเซลต่อหนึ่งหน่วยความยาวเรียกว่าเรสโซรูชัน (resolution) ซึ่งมักจะกำหนดเป็นจำนวนพิกเซลต่อนิ้วในระบบอเมริกัน หรือพิกเซลต่อมิลลิเมตร ในระบบอื่น ในระบบที่มีเรสโซรู

ชันสูงๆนั้นการปัดเศษของพิกเซลจะถือได้ว่าไม่มีผลต่อรูปร่างของภาพเลย แต่อย่างไรก็ตาม ถ้าเป็นในระบบที่มีเรสโซรูชันต่ำ การปัดเศษของพกเซลทิ้งไปหรือปัดขึ้นอาจ ทำให้รูปร่างของภาพที่ต้องการนั้นเปลี่ยนไปได้

2.2 *การสร้างเส้นตรงหรือเส้นโค้ง*

วิธีการสร้างเส้นตรงของเมตาฟอนต์นั้นทำได้ง่ายๆโดยการกำหนดจุดสองจุดบนระบบอ้างอิงและใช้คำสั่ง draw ระหว่างสองจุดนั้น เมตาฟอนต์จะดำเนินการลากเส้นตรงให้ทันที รูปแบบคำสั่งจะเป็นดังนี้

draw (x1,y1)..(x2,y2)

จะเป็นการลากเส้นตรงระหว่างจุด (x1,y1) ไปยังจุด (x2,y2) เป็นต้น

วิธีการสร้างเส้นโค้งต่างของเมตาฟอนต์จะมีวิธีการที่สลับซับซ้อนอยู่บ้างซึ่งสามารถอธิบายโดยยกตัวอย่างประกอบได้ดังนี้

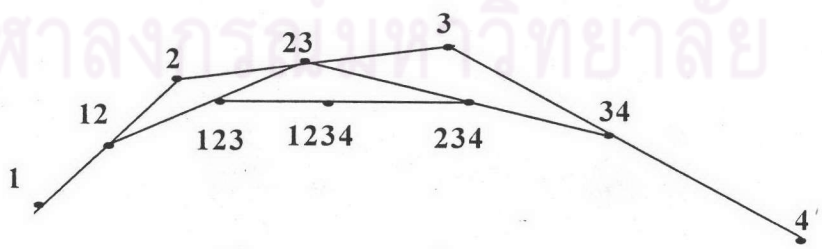
สมมุติว่าตอนเริ่มต้นมีจุดพิกัดอยู่ 4 จุดคือ z1,z2,z3,z4 จากนั้นจะมีการ กำหนดจุดที่เป็นกึ่งกลางได้ 3 จุดคือ

z12 = 1/2[z1,z2],

z23 = 1/2[z2,z3],

z34 = 1/2[z3,z4]

ดังที่แสดงในรูปที่ 2.5



รูปที่ 2.5 แสดงการจุดกึ่งกลางบนเส้นตรง

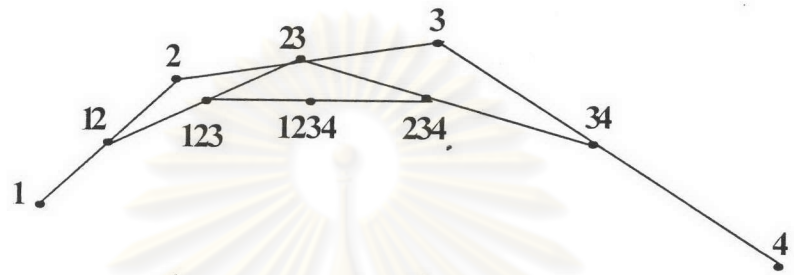
หลังจากนั้นสร้างจุดกึ่งกลางลำดับที่สองของจุดกึ่งกลางที่ได้ในขั้นแรก (z12,z23,z34) จะได้

z123 = 1/2[z12,z23],

z234 = 1/2[z23,z34]

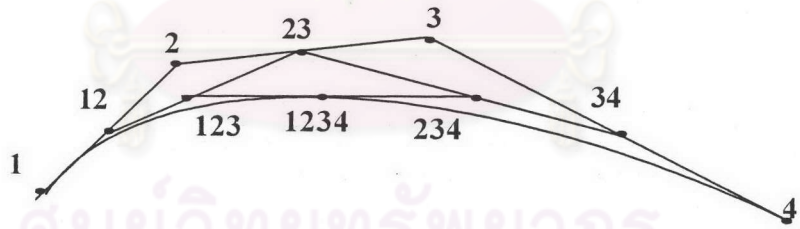
หลังจากนั้นสร้างจุดกึ่งกลางลำดับที่สาม จะได้

z1234 = 1/2[z123,z234] ดังรูปที่ 2.6



รูปที่ 2.6 แสดงการแบ่งจุดกึ่งกลางของเส้นตรงย่อย

จุด z1234 ที่ได้นี้จะเป็นจุดหนึ่งที่อยู่บนเส้นโค้งที่กำหนดโดย (z1,z2,z3,z4) สำหรับการหาจุดอื่นๆที่เหลือเพื่อที่จะสามารถทำการลากเส้นโค้งนี้ได้นั้น ก็จะใช้วิธีการเช่นเดียวกัน โดยจะเป็นการหาแนวเส้นโค้งที่กำหนดโดย (z1,z12,z123,z1234) และ (z1234,z234,z34,z4) จะทำเช่นนี้ไปเรื่อยๆ ในที่สุดขั้นตอนการประมวลผลก็จะสิ้นสุดลงโดยรวดเร็วและก็จะได้แนวเส้นโค้งที่ต้องการดังแสดงในรูปที่ 2.7



รูปที่ 2.7 แสดงส่วนโค้งที่ได้จากการทำงานของเมตาฟอนต์

คุณสมบัติที่สำคัญของเส้นโค้งที่ได้จะมีดังนี้

-จะเริ่มต้นที่ z1 และมีทิศทางจาก z1 ไปยัง z2

-จะสิ้นสุดที่ z4 และมีทิศทางจาก z3 ไปยัง z4

-แนวเส้นโค้งทั้งหมดจะอยู่ภายใต้กรอบของยอด z1,z2,z3,z4

การสร้างเส้นโค้งที่กำหนดโดยวิธีการเรียกตัวเองดังที่กล่าวข้างต้นนี้สามารถ สรุปเป็นสมการได้ดังนี้

$$z(t) = (1-t)^3 z1 + 3(1-t)^2 tz2 + 3(1-t)t^2 z3 + t^3 z4$$

โดย t คือพารามิเตอร์ที่มีค่าตั้งแต่ 0 ถึง 1 สมการโพลีโนเมียลที่มีดีกรีของ t เป็น 3 นี้เรียกว่า Bernshtein polynomial

2.3 *การขยายขนาดและความละเอียด (Magnification and Resolution)*

จุดเด่นอย่างหนึ่งของเมตาฟอนต์ก็คือสามารถที่จะผลิตฟอนต์สำหรับอุปกรณ์ การพิมพ์หลายๆ ชนิด ถ้าผู้ออกแบบโปรแกรมได้ออกแบบให้เรสโซรูชันเปลี่ยนแปลงค่าได้

สมมุติว่ามีอุปกรณ์เอาท์พุตอยู่ 2 ชนิด ชนิดแรกให้ชื่อว่าชนิด dot มีความละเอียด 200 พิกเซลต่อนิ้ว (ประมาณ 8 พิกเซลต่อมิลลิเมตร) อีกชนิดหนึ่งมีชื่อว่า laser มีความละเอียด 2000 พิกเซลต่อนิ้ว ต้องการสร้างโปรแกรมเมตฟอนต์ที่สามารถแสดงผลได้กับอุปกรณ์ทั้งสองประเภท ดังนั้นถ้าจำเป็นต้องพิมพ์ผลลัพธ์ของแฟ้มที่มีชื่อว่า test.mf ออกทางเครื่อง dot ก็เพียงแต่ใช้คำสั่ง

```
\mode=dot; input test
```

หรือถ้าต้องการพิมพ์ออกทางเครื่องพิมพ์ที่มีคุณภาพสูงขึ้นเพียงแต่เปลี่ยนจาก dot เป็น laser เท่านั้น

```
\mode=laser; input test
```

ด้วยวิธีการดังที่กล่าวมานี้ ถ้าต้องการพิมพ์ผลลัพธ์ออกทางอุปกรณ์เอาท์พุตอื่นๆ จะสามารถทำได้โดยเปลี่ยนค่าตัวแปรของโหมด (mode variable) เท่านั้น

วิธีการทำงานข้างต้นนี้สามารถทำได้โดยใส่คำสั่ง mode_setup ไว้ที่ตอนเริ่มต้นของโปรแกรม test และคำสั่งที่ใส่ไว้นี้จะไปทำการกำหนดค่ากลุ่มของตัวแปรกลุ่มหนึ่งที่ทำหน้าที่ควบคุมการแสดงผลลัพธ์ซึ่งจะมีส่วนเกี่ยวพันกับจำนวนของพิกเซลต่อหน่วยความยาว จากตัวอย่างข้างต้น เมื่อโหมดถูกกำหนดให้เป็น dot ค่าของ pt จะเท่ากับ 2.7674 พิกเซล (pt คือจำนวนจุดของเครื่องพิมพ์ (printer's point) ซึ่งเป็นหน่วยชนิดหนึ่งเช่นเดียวกับมิลลิเมตร 72.27 pt = 1 นิ้ว)และ mm = 7.87402 พิกเซล แต่เมื่อโหมดถูกกำหนดให้เป็น laser ค่าของ pt = 27.674 และ mm = 78.4017 โปรแกรม test.mf ที่ทำงานภายใต้โหมด dot จะให้ผลลัพธ์ที่มีขนาดเล็กและสั้นกว่าที่ทำงานภายใต้โหมด laser ประมาณ 10 เท่า หรือในกรณีที่ต้องการเขียนเส้นตรงให้มีขนาดความยาว 3 มิลลิเมตร จากพิกัด (0,0) ไปยัง (3mm,0) ในโหมด dot จะใช้จำนวนพิกเซลเท่ากับ 23.6 พิกเซล และโหมด laser จะใช้จำนวนพิกเซลเท่ากับ 236.2 พิกเซล

ความยุ่งยากซับซ้อนที่เกิดขึ้นของการพิมพ์อีกอย่างหนึ่งก็คือการขยายขนาด หรือย่อส่วนของผลลัพธ์ซึ่งไม่ใช่ขนาดเดียวกับที่ทำการออกแบบไว้ตอนเริ่มต้นเช่น ถ้าต้องการให้ผลลัพธ์ที่ได้ภายใต้โหมด dot มีขนาดใหญ่ขึ้นเป็นสองเท่า เท็กซ์จะใช้คำสั่ง

```
\magnification=2000
```

ในการทำให้ขนาดของเอาท์พุตทั้งหมดใหญ่ขึ้นเป็นสองเท่า หรืออาจจะใช้คำสั่ง

```
\font\f=test scaled 2000
```

สำหรับทำให้ฟอนต์ภายใต้โปรแกรม test.mf เท่านั้นมีขนาดใหญ่ขึ้นเป็นสองเท่า วิธีการจัดเตรียมฟอนต์ของโปรแกรมเมตาฟอนต์เพื่อให้ใช้งานได้กับเท็กซ์สามารถทำได้โดยใช้คำสั่ง

```
\mode=dot; mag=2; input test;
```

ซึ่งจะทำให้ขนาดของ pt = 5.5348 พิกเซล และ mm = 15.74803 พิกเซล

การทำงานของรูทีน mode_setup จะทำการตรวจสอบว่าค่าของ mag เป็นเท่าใด ถ้าไม่มีการกำหนดค่าของ mag จะถือว่ามีค่าเป็น 1 ในทำนองเดียวกัน ถ้าไม่มีการกำหนดชนิดของโหมดที่ใช้ เมตาฟอนต์จะถือว่า mode = proof

หน่วยที่เมตาฟอนต์ใช้ในการอ้างอิงต่างๆมีดังนี้

pt	printer's point	(72.27 pt = 1 in)
pc	pica	(1 pc = 12 pt)
in	inch	(1 in = 2.54 cm)
bp	big point	(72 bp = 1 in)
cm	centimeter	(100 cm = 1 meter)
mm	millimeter	(10 mm = 1 cm)
dd	didot point	(1157 dd = 1238 pt)
cc	cicero	(1 cc = 12 dd)

ในแต่ละหน่วยนั้นถ้ามีการปัดเศษของหน่วยจากการคำนวณ เมตาฟอนต์จะทำการปัดให้ใกล้เคียงกับ 1/65536 ส่วนของพิกเซล

หน่วยของพิกเซลที่กล่าวถึงข้างต้นนั้นเรียกได้ว่าเป็นมิติทางกายภาพ (physical dimension) ซึ่งเมตาฟอนต์จะไม่ค่อยนิยมใช้เท่าใดนักเนื่องจากจะเป็นการทำให้การเขียนโปรแกรมไม่มีความยืดหยุ่น เมตาฟอนต์จะมีหน่วยสำหรับบอกมิติอีกประเภทหนึ่งซึ่งจะให้ความสะดวกในการออกแบบฟอนต์มากกว่าเช่น จะนิยมใช้ 'em' หรือ 'x_height' ในการกำหนดขนาดของตัวอักษร ซึ่งจะแปรเปลี่ยนไปตามชนิดของฟอนต์ เป็นต้น

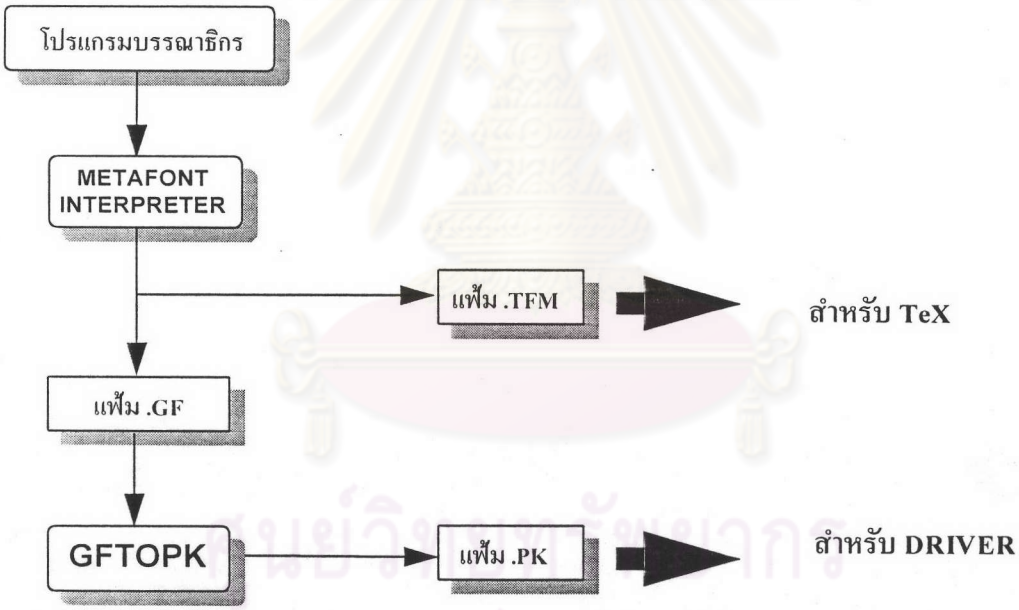
เนื่องจากการขยายขนาดหรือการกำหนดความละเอียดในโหมดต่างๆจะมีผลทำให้จำนวนของพิกเซลต่อหน่วยมิติทางกายภาพนี้เปลี่ยนไปด้วย วิธีการที่จะให้จำนวนหน่วยของพิกเซลมีค่าคงที่ไม่เปลี่ยนแปลงไปตามการกำหนดความละเอียดในโหมดต่างๆนั้น เมตาฟอนต์จะกำหนดปริมาณชาร์ป(sharp quantity) ขึ้นเพื่อ

ใช้ในการนี้ โดยเติมเครื่องหมาย '#' ต่อท้ายหน่วยที่ต้องการก็จะทำให้ได้หน่วยของมิติที่ไม่เปลี่ยนแปลงค่าเมื่อมีการเปลี่ยนแปลงความละเอียด

หลักการที่กล่าวมาข้างต้นนี้เป็นหลักการที่จำเป็นในการที่จะพัฒนาโปรแกรมสร้างฟอนต์ด้วยเมตาฟอนต์ อย่างไรก็ตามขั้นตอนในการพัฒนาโปรแกรมด้วยเมตาฟอนต์สามารถ สรุปได้ดังนี้

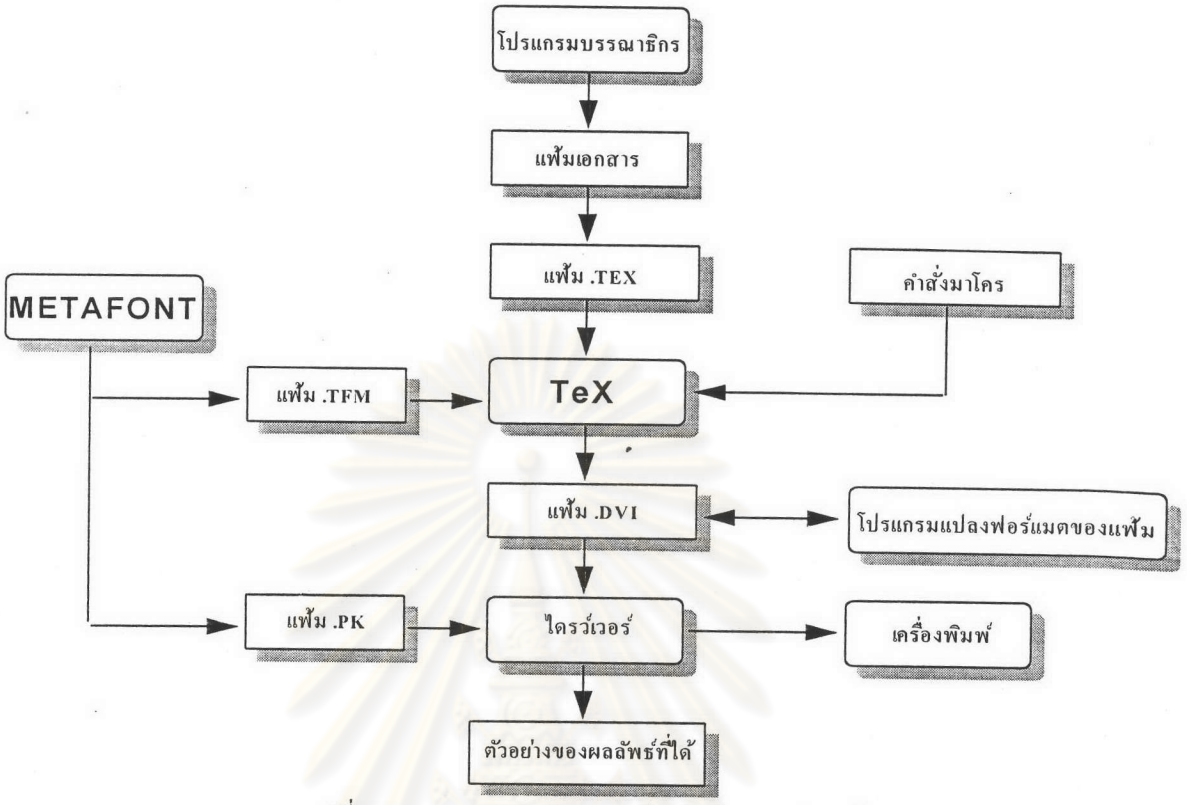
1. เขียนโปรแกรมด้วยชุดคำสั่งของเมตาฟอนต์จะได้แฟ้มที่มีนามสกุล .mf
2. ใช้เมตาฟอนต์ประมวลผลแฟ้ม .mf จะได้แฟ้มที่มีนามสกุล .tfm และ .gf (หรือ .xxxgf ขึ้นอยู่กับระบบปฏิบัติการที่ใช้)
3. เปลี่ยนแฟ้มที่มีนามสกุล .gf ให้เป็นแฟ้มที่มีนามสกุล .pk หรือ .pxl ทั้งนี้ขึ้นอยู่กับซอฟท์แวร์ไดรว์เวอร์ที่เท็กซ์ต้องการ

ขั้นตอนการทำงานของโปรแกรมเมตาฟอนต์สามารถแสดงได้ดังรูปที่ 2.8



รูปที่ 2.8 แสดงการทำงานโดยทั่วไปของเมตาฟอนต์

เมื่อนำมาประกอบกับขั้นตอนการทำงานของเท็กซ์แล้วจะเป็นดังรูปที่ 2.9



รูปที่ 2.9 แสดงการทำงานของเท็กซ์และเมตาฟอนต์

3. การตัดคำ

สิ่งที่มักจะพบกันอยู่เสมอในการประมวลผลคำด้วยโปรแกรมประมวลผลคำหรือเวิร์ดโปรเซสเซอร์ที่นิยมใช้กันโดยทั่วไปก็คือการตัดหรือแบ่งคำในการขึ้นบรรทัดใหม่ปัญหาที่กล่าวมานี้ เป็นตัวแปรสำคัญอย่างยิ่งที่ทำให้ความน่าใช้ของโปรแกรมประมวลผลคำลดลงอย่างมากทีเดียว ในบางครั้งจะพบว่าโปรแกรมประมวลผลคำนั้นมีฟอนต์ที่สวยงามให้เลือกใช้อย่างหลากหลายหรือมีรูทีนในการที่จะแสดงผลให้ผู้ใช้สามารถมองเห็นได้ก่อนที่จะทำการพิมพ์งานลงบนกระดาษก็ตาม ถ้าอัลกอริธึมในการตัดคำของโปรแกรมนั้นไม่ดีเพียงพอ หรือเมื่อทำการเว้นวรรคแล้วทำให้ช่องว่างระหว่างตัวอักษรหรือช่องว่างระหว่างคำมากเกินไป ก็จะทำให้ความน่าสนใจของโปรแกรมนั้นลดลงอย่างมากเลยทีเดียว การออกแบบอัลกอริธึมสำหรับการตัดคำนี้เป็นปัญหาที่ยากยิ่งในหลายๆภาษา รวมทั้งภาษาไทยด้วยเช่นกันในตัวโปรแกรมของเท็กซ์เองก็มีอัลกอริธึมสำหรับจัดการในส่วนนี้ด้วย แต่เนื่องจากการตัดคำของเท็กซ์นั้นทำไว้สำหรับภาษาอังกฤษ โดยโครงสร้างทางภาษาแล้วภาษาอังกฤษจะมีลักษณะเป็นคำๆ ในหนึ่งประโยคจะประกอบด้วยคำต่างๆมา

ประกอบกันและมีการแบ่งแยกระหว่างคำด้วยวรรคหรือช่องว่าง ซึ่งแตกต่างจากภาษาไทยซึ่งในหนึ่งประโยคจะนำคำมาเรียงติดกัน ไปหมดไม่มีการใช้วรรคหรือช่องว่างในการแบ่งจนกว่าจะจบประโยคหรือเริ่มต้นเรื่องใหม่ ดังนั้น การพัฒนาอัลกอริธึมที่ใช้ในโปรแกรมเท็กซ์ให้ใช้ได้กับการตัดคำภาษาไทยนั้นค่อนข้างที่จะยุ่งยากคง ต้องใช้ระยะเวลาในการศึกษากันต่อไป อย่างไรก็ตามสำหรับการตัดคำภาษาไทยเองนั้นมี อัลกอริธึมที่ใช้กันอยู่มากมาย และที่จะนำมาใช้ในการพัฒนาโปรแกรมไทยเท็กซ์นี้จะเลือกใช้อัลกอริธึมตัดคำที่ใช้อยู่ในโปรแกรม CU Writer

ถึงแม้ว่าอัลกอริธึมตัดคำที่มีอยู่ในโปรแกรมเท็กซ์นั้นจะยังไม่สามารถนำมาใช้ได้กับโปรแกรมไทยเท็กซ์ก็ตาม แต่วิธีการที่อัลกอริธึมนี้ใช้ก็เป็นวิธีที่น่าสนใจมากซึ่งสามารถสรุปได้ดังนี้

3.1 *การตัดคำด้วยวิธี Hyphenation ที่ใช้ในโปรแกรมเท็กซ์*

เท็กซ์จะทำการตัดคำที่ให้มาโดยเริ่มต้นเปรียบเทียบกับคำที่มีใน "exception dictionary" หรือพจนานุกรมเก็บคำเฉพาะที่ต้องกำหนดวิธีแบ่งเอาไว้เป็นพิเศษ ถ้าคำที่ต้องการจะตัดตรงกับคำเฉพาะนั้นก็จะใช้วิธีการตัดคำแบบพิเศษนั้นเลย ถ้าไม่พบก็จะทำการค้นหารูปแบบ (pattern) ของคำนั้นและนำมาเปรียบเทียบกับรูปแบบ (pattern) ที่เท็กซ์มีอยู่ เพื่อที่จะให้เข้าใจง่ายขึ้นจำเป็นต้องยกตัวอย่างประกอบโดยใช้คำว่า " hyphenation"

เมื่อเริ่มต้นเท็กซ์จะทำการใส่จุด "." เข้าไปวางไว้หน้าคำและหลังคำแห่ง ละหนึ่งจุด

.hyphenation.

จะถือว่าคำ ".hyphenation." นี้เป็นคำใหม่ที่จะดำเนินการค้นหาต่อไป ถ้านำเอาตัวอักษรที่ประกอบเป็นคำใหม่นี้มาแยกเป็นคำย่อยที่มีความยาวของตัวอักษรในแต่ละคำเท่ากับ 1 จะได้

. h y p h e n a t i o n .

ถ้าให้ความยาวของตัวอักษรในแต่ละคำเท่ากับ 2 จะได้

.h hy yp ph he en na at ti io on n.

ถ้าให้ความยาวของตัวอักษรในแต่ละคำเท่ากับ 3 จะได้

.hy hyp yph phe hen ena nat ati tio ion on.

กระทำเช่นนี้ไปเรื่อยๆจนความยาวของคำย่อยที่ได้เท่ากับคำเริ่มต้น

จากวิธีการข้างต้นนี้จะพบว่าแต่ละคำย่อยที่มีความยาวของตัวอักษรเท่ากับ k ตัว จะสามารถใส่ตัวเลขห้อยข้างหน้าตัวอักษรและข้างท้ายตัวอักษรได้ k+1 ตัวสำหรับตำแหน่ง ของตัวอักษรที่ติดกันเช่นคำย่อย " hen" สามารถใส่ตัวเลขได้เป็น $_0h_0e_2n_0$ ตัวเลข 2 ที่เห็นนี้จะหมายถึงตำแหน่งที่อาจจะแบ่งคำโดยใส่เครื่องหมายไฮเฟนได้ วิธีที่จะรู้ว่าคำย่อยแต่ละคำมีตัวเลขอะไรบ้างนั้นจะทำได้โดยการค้นหาใน "pattern dictionary" หรือพจนานุกรมที่เก็บ รูปแบบของคำ (ในระบบเท็กซ์พจนานุกรมดังกล่าวจะถูกเก็บไว้ในแฟ้มที่ชื่อว่า "hyphen.tex") ถ้าหาไม่พบจะถือว่าตัวเลขที่ห้อยอยู่ระหว่างตัวอักษรนั้นมีค่าเป็นศูนย์ทั้งหมด ในที่นี้เราจะนำเฉพาะคำที่หาพบออกมาเพื่อเปรียบเทียบกันต่อไป ซึ่งรูปแบบที่สามารถหาพบคือ

$_0h_0y_3p_0h_0$ $_0h_0e_2n_0$ $_0h_0e_0n_0a_4$ $_0h_0e_0n_5a_0t_0$ $_1n_0a_0$

$_0n_2a_0t_0$ $_1t_0i_0o_0$ $_2i_0o_0$

หลังจากนั้นก็จะทำการคำนวณหาค่าตัวเลขที่ห้อยอยู่ระหว่างตัวอักษรที่มีค่ามากที่สุดสำหรับตัวอักษรคู่เดียวกันที่อยู่ในรูปแบบที่ต่างกัน เช่นในระหว่างตัวอักษร "e" และ "n" จะมีรูปแบบที่เกี่ยวข้องอยู่ทั้งหมด 4 รูปแบบ ดังนั้นจะมีตัวเลขที่จะนำมาเปรียบเทียบอยู่ 4 ตัว (2 จาก $_0h_0e_2n_0$, 0 จาก $_0h_0e_0n_0a_4$, 0 จาก $_0h_0e_0n_5a_0t_0$ และ 1 จาก $_1n_0a_0$) ตัวเลขที่มีค่ามากที่สุดสำหรับกรณีนี้คือ 2 เมื่อใช้วิธีการนี้ทำจนครบทุกรูปแบบแล้วจะได้ผลลัพธ์คือ

$._0h_0y_3p_0h_0e_2n_5a_4t_2i_0o_0n_0.$

ขั้นตอนสุดท้ายคือการแบ่งคำด้วยไฮเฟน วิธีการพิจารณาการแบ่งคำนี้จะเลือกเฉพาะตำแหน่งที่เป็นตัวเลขคี่เท่านั้น ในตัวอย่างนี้จะสามารถแบ่งได้สองตำแหน่งคือ

hy-phen-ation

ในการทำงานของเท็กซ์ รูปแบบของคำเหล่านี้ซึ่งมีประมาณ 4447 รูปแบบจะถูกโหลดลงไว้ในหน่วยความจำของเท็กซ์ ค่าตัวเลขที่ห้อยอยู่จะมีค่าสูงสุดเป็น 5 ซึ่งหมายความว่าในตำแหน่งนั้นจะสามารถใช้เป็นจุดแบ่งคำได้อย่างแน่นอน อัลกอริธึมตัดคำที่เท็กซ์ใช้นี้เรียกว่า Liang's algorithm ข้อดีของอัลกอริธึมนี้คือไม่จำเป็นต้องใช้คำทั้งหมดที่มีอยู่ในพจนานุกรมมาเป็นตัวเปรียบเทียบ จึงทำให้สามารถประหยัดเนื้อที่ของหน่วยความจำที่ใช้เป็นอันมาก และไม่จำเป็นต้องเปลี่ยนพจนานุกรมใหม่ทุกครั้งที่มีคำใหม่เพิ่มขึ้น เนื่องจากพจนานุกรมรูปแบบที่มีอยู่จะบ่งบอกถึงไวยากรณ์และโครงสร้างของการผสมตัวอักษรให้เป็นคำอยู่ในตัวเองอยู่แล้ว ถ้ามีคำใหม่ซึ่งต้องการแบ่งด้วยรูปแบบที่พิเศษไปกว่านี้ก็จะสามารถนำไปบรรจุไว้ใน "exception dictionary" แทน

3.2 *การตัดคำด้วยอัลกอริธึมของโปรแกรม CU Writer*

การทำงานของอัลกอริธึมตัดคำหรือแบ่งคำของโปรแกรม CU Writer นี้ จะใช้พจนานุกรมมาเป็นเกณฑ์ในการแบ่ง แต่วิธีการเก็บพจนานุกรมนั้นจะไม่ใช้การเก็บในลักษณะของคำต่อคำเนื่องจากจะทำให้เสียเนื้อที่หน่วยความจำเป็นจำนวนมาก และจะใช้เวลาในการค้นหานาน ดังนั้นจึงจำเป็นที่จะต้องมีการสร้างรูปแบบขึ้นมาเพื่อจัดการเปลี่ยนลักษณะพจนานุกรมที่เก็บคำศัพท์ชนิดคำต่อคำให้เป็นรูปแบบที่ง่ายในการค้นหาและสิ้นเปลืองหน่วยความจำน้อยลง

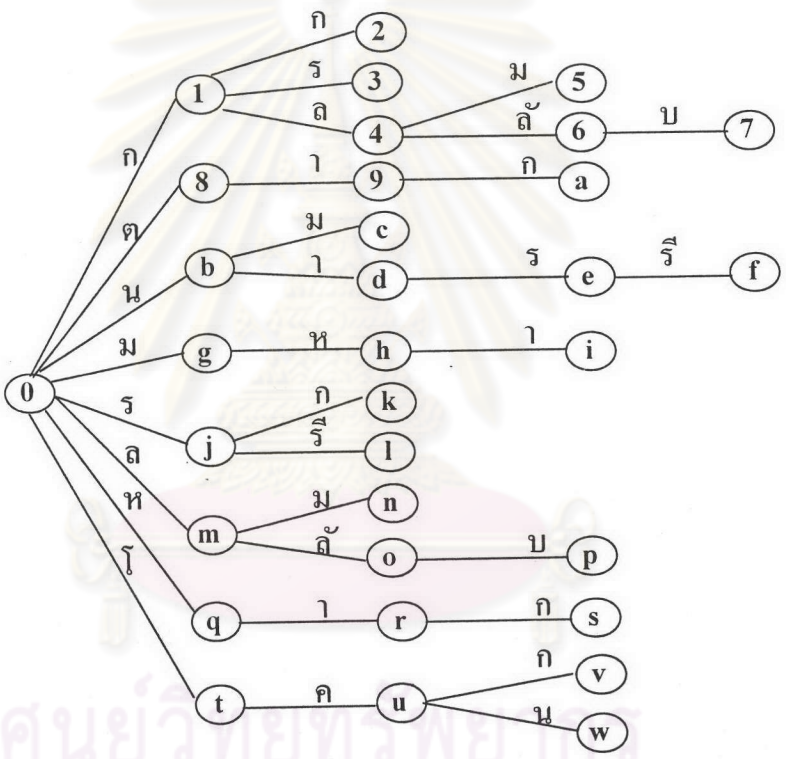
3.2.1 การแทนพจนานุกรมในหน่วยความจำด้วยโครงสร้างข้อมูลแบบทรัย (Trie)

คำว่าพจนานุกรมในความหมายของอัลกอริธึมตัดคำนั้นจะหมายถึงโครงสร้างของข้อมูลแบบต้นไม้ (tree structure) โครงสร้างของข้อมูลแบบต้นไม้หรือทรี มีองค์ประกอบหลักคือ สถานะ (state) และทางเลือกที่จะไปยังสถานะใหม่เรียกว่าทรานสิชั่น (transition) วิธีการที่จะทำความเข้าใจเกี่ยวกับการแทนที่พจนานุกรมด้วยทรีให้ดียิ่งขึ้นนั้นจำเป็น ต้องอาศัยตัวอย่างประกอบดังนี้

สมมุติว่าในพจนานุกรมของภาษาไทยประกอบด้วยคำจำนวน 20 คำคือ

กก กร กล กลม กลับ ตา ตาก นม นา นารี มหา รก รี ลม ลับ หา หาก โค โคก โคน

ดังนั้นประโยคหรือวรรคในภาษาไทยก็คือสตริงของอักขระที่ประกอบด้วยคำใดๆก็ตามที่อยู่ในพจนานุกรมนี้มาเรียงต่อกันนั่นเอง สำหรับการนำคำดังกล่าวมาสร้างเป็นทรีนั้น จะสามารถทำได้ดังรูปที่ 2.10

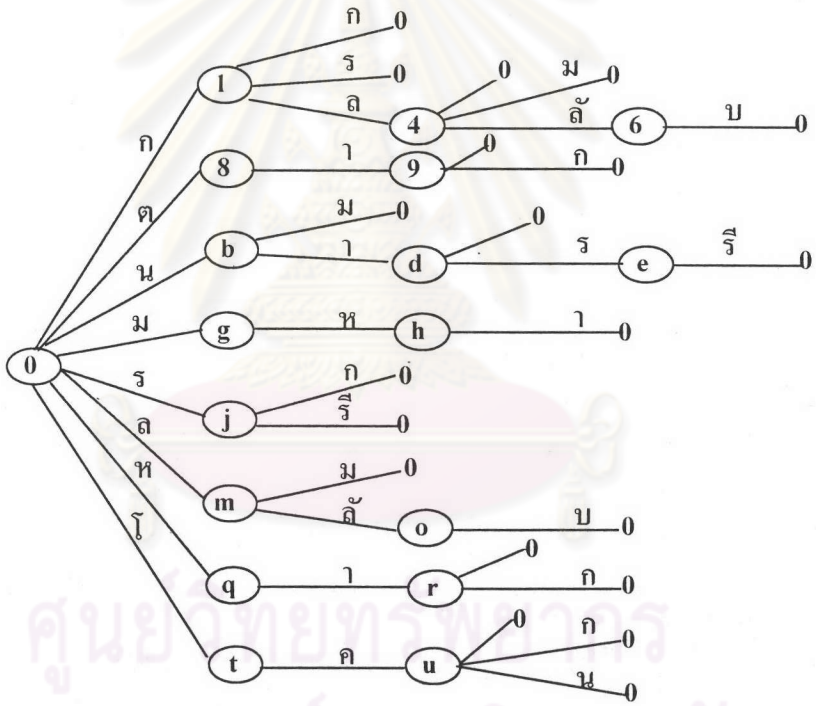


รูปที่ 2.10 แสดงแผนภาพการใช้ DFA เก็บพจนานุกรม

สิ่งที่ได้นี้เรียกว่าทรัย (trie) หรือ deterministic finite state automaton (DFA) ของพจนานุกรมวิธีการทำงานของ DFA นี้จะเริ่มต้นที่สถานะ 0 จากนั้นจะรับตัวอักษรที่ป้อนเข้ามาทีละตัวและเดินไปตามเส้นทางต่างๆ ตามตัวอักษรที่ได้รับ ถ้าตัวอักษรที่ได้รับในแต่ละสถานะนั้นแตกต่างไปจากทางเลือกที่มีอยู่ก็แสดงว่าสตริงของอักขระนั้นไม่มีอยู่ในพจนานุกรมนี้ในทางกลับกันถ้าตัวอักษรที่ได้รับในทุกสถานะสามารถทำให้เดินไป

ถึงสถานะสุดท้ายที่เป็นสถานะสิ้นสุด (final state)ได้โดย ไม่เกิดข้อผิดพลาดขึ้นเสียก่อน แสดงว่าสตริงของอักขระนั้นเป็นคำที่มีอยู่ในพจนานุกรม (สถานะสุดท้ายหมายถึงสถานะที่ไม่มีทางเดินต่อไปอีกแล้ว) สิ่งที่กล่าวมาข้างต้นนี้ แท้จริงแล้วคือการตรวจสอบความถูกต้องของตัวสะกดของคำที่ใช้กันอยู่ในโปรแกรมประมวลผลคำทั่วๆไป

ถ้าต้องการจะค้นหาจุดสำหรับแบ่งคำในประโยคหรือวรรค จะมีความแตกต่างออกไปบ้างเล็กน้อย เนื่องจากอัลกอริธึมในการตัดคำนั้นไม่ได้ต้องการแค่เพียงว่าคำนั้นอยู่ในพจนานุกรมหรือไม่ แต่ต้องการหาจุดที่เหมาะสมในการแบ่งด้วย ดังนั้นจึงต้องมีการสร้าง nondeterministic finite state automaton (NFA) ของประโยคหรือวรรคแทน ซึ่งจาก DFA ของพจนานุกรมสามารถเปลี่ยนเป็น NFA ของประโยคหรือวรรคได้ดังรูปที่ 2.11



รูปที่ 2.11 แสดงแผนภาพการสร้าง NFA

ขั้นตอนในการทำงานของการแบ่งคำนี้ก็จะคล้ายคลึงกับการตรวจสอบความถูกต้องของตัวสะกดของคำดังที่กล่าวมาแล้วข้างต้น จะมีข้อแตกต่างกันอยู่ตรงที่เมื่อเดินมาตามทางจนถึงสถานะสุดท้ายที่ไม่มีทางไปต่ออีกแล้วนั้น ในการแบ่งคำนี้จะไม่มีสถานะสุดท้ายแต่จะเป็นสถานะ 0 แทน สังเกตพบว่าเป็นสถานะเดียวกันกับสถานะเริ่มต้น ซึ่งแปลความหมายได้ว่าความยาวของคำๆนี้สิ้นสุดลงแล้วมีทางเลือกว่าจะสิ้นสุดประโยคหรือ

วรรคตรงนี้ หรืออาจจะเริ่มต้นคำใหม่ต่อไปก็ได้ แต่บางครั้งก็อาจจะเดินมาถึงสถานะสิ้นสุด (final state) ที่ไม่ใช่สถานะสุดท้าย (วงกลมล้อมรอบสถานะสิ้นสุดจะมีความเข้มมากกว่าปกติ) ถ้าเป็นเช่นนี้จะหมายความว่าอาจจะหยุดคำลงที่ตรงตำแหน่งนี้ หรือจะสิ้นสุดประโยคเพียงเท่านี้ หรืออาจจะรอการป้อนตัวอักษรต่อไปเพื่อนำมาประกอบเป็นคำที่ยาวขึ้นก็เป็นได้

ความยุ่งยากที่ตามมาจากการมีทางเลือกมากกว่าหนึ่งทางก็คือทางไหนคือทางที่ถูกต้อง คำตอบสำหรับคำถามนี้ก็คือทางที่เดินไปแล้วสำเร็จ (successful path) เส้นทางที่เรียกได้ว่าเป็นเส้นทางที่สำเร็จก็คือเส้นทางที่เดินไปแล้วพบสถานะสิ้นสุดมีค่าเป็น 0 ตามหลักการแล้วการที่จะยอมรับว่าประโยคหรือวรรคนั้นสิ้นสุดหรือไม่จะต้องพิจารณาว่าต้องมีเส้นทาง ที่เดินไปแล้วสำเร็จอย่างน้อย1 เส้นทาง นับจากสถานะเริ่มต้น (start state) ไปยังสถานะสิ้นสุด (final state)

ตัวอย่างที่สามารถยกมาอธิบายได้เช่น จากประโยค "หากโคกล" เมื่อนำมาป้อนเข้ากับ NFA จะมีเส้นทางที่เป็นไปได้อยู่ 3 เส้นทางคือ

0--ห--q--า--0--ก--1--โ

0--ห--q--า--r--ก--0--โ--t--ค--0--ก--1--ล--0

0--ห--q--า--r--ก--0--โ--t--ค--u--ก--0--ล--i

เส้นทางที่ถือว่าสำเร็จคือเส้นทางที่ 2 เพราะจบลงด้วยสถานะสิ้นสุดเป็น 0 เส้นทางที่ 1 นั้นเรียกว่าเป็นการล้มเหลวเนื่องจากไม่สามารถรับตัวอักษรต่อไปได้ สำหรับเส้นทางที่ 3 ถึงแม้ว่าจะรับตัวอักษรได้จนจบแต่ก็ไม่ถือว่าสำเร็จเพราะในสถานะนั้นยังต้องรอตัวอักษรที่จะต้องนำมาประกอบเพื่อให้สถานะนั้นสมบูรณ์ต่อไป

เมื่อได้เส้นทางที่ถือว่าสำเร็จแล้ว ก็จะนำมาพิจารณาหาจุดที่แบ่งคำ จากตัวอย่างข้างต้นเราได้เส้นทางที่สำเร็จคือ

0--ห--q--า--r--ก--0--โ--t--ค--0--ก--1--ล--0

ซึ่งเมื่อพิจารณาแล้วจะพบว่าคำที่อยู่ระหว่างสถานะ 0 จะเป็นคำที่มีอยู่ในพจนานุกรม 20 คำที่กำหนดไว้ตั้งแต่ต้น ดังนั้นจึงสรุปได้ว่าสถานะสิ้นสุดที่มีค่าเป็นศูนย์ก็คือจุดที่ใช้แบ่งคำนั่นเอง การแบ่งคำที่ได้จึงเป็น (หาก)(โค)(กล)

3.2.2 การแบ่งคำด้วยทรัย

จากที่อธิบายไว้ข้างต้นจะพบว่าการแบ่งคำจะใช้หลักการของ Finite state Automaton ซึ่งก็คือโครงสร้างของทรัย ดังนั้นอัลกอริธึมแบ่งคำที่ใช้วิธีการทำงานของทรัยจะสามารถเข้ากันได้ดีกับงานประเภทนี้ สิ่งที่น่าสนใจต่อไปก็คือสถานะต่างๆที่ได้จากทรัยจะมีโอกาสเป็นไปได้ 4 แบบคือ

1. สถานะที่ยังไม่สามารถแบ่งคำได้ คือ สถานะที่ไม่ใช่สถานะสิ้นสุด
2. สถานะที่สามารถแบ่งคำได้ คือสถานะนั้นเป็นสถานะสิ้นสุดแล้ว แต่ยังมีเส้นทางต่อไปอีก ในสถานะนี้จึงมีทางเลือกว่าจะแบ่งหรือไม่แบ่งก็ได้
3. สถานะที่บังคับให้ต้องแบ่งคำ คือสถานะสิ้นสุดที่ไม่มีเส้นทางต่อไปอีกแล้ว
4. สถานะที่ไม่สำเร็จ คือสถานะที่มีความผิดพลาดเนื่องจากตัวอักษรที่ได้รับไม่อยู่ในเส้นทางที่ทรัยมีอยู่

ปัญหาต่อไปก็คืออะไรจะเป็นตัวที่บ่งชี้ได้ว่าสตริงของอักขระที่ป้อน เข้าไปนั้นอยู่ในสถานะใด วิธีการตรวจสอบสถานะนี้จะใช้หลักการคือ จะให้ s แทนสตริงของอักขระ current แทนคำที่กำลังค้นหา และให้ next แทนคำที่อยู่ในลำดับต่อจาก current (ในที่นี้มีสมมุติฐานว่าพจนานุกรมถูกเรียงตามลำดับแล้ว โดยการเรียงลำดับนี้ไม่จำเป็นต้องเป็นการเรียงตามอักขระวิธีที่ใช้ในพจนานุกรมฉบับราชบัณฑิตยสถาน) จากนั้นจะพิจารณาได้ดังนี้

1. สถานะที่ไม่สำเร็จ

ถ้าเปรียบเทียบตัวอักษรที่อยู่ในs ด้วยจำนวนตัวอักษรที่เท่ากับที่มีอยู่ใน current แล้วพบว่าไม่เท่ากัน แสดงว่าการค้นหานี้ล้มเหลว จะเรียกว่าสถานะที่ไม่สำเร็จ

2. สถานะที่ไม่สามารถแบ่งคำได้

ถ้าเปรียบเทียบตัวอักษรที่อยู่ในs ด้วยจำนวนตัวอักษรเท่ากับที่มีอยู่ใน current แล้วพบว่าเท่ากัน แต่ตัวอักษรที่มีอยู่ใน s มีมากกว่าตัวอักษรที่มีอยู่ใน current แสดงว่า ยังไม่เป็นคำ

3. สถานะที่สามารถแบ่งคำได้

ถ้าเปรียบเทียบตัวอักษรที่อยู่ใน s ด้วยจำนวนตัวอักษรเท่ากับที่มีอยู่ใน current แล้วพบว่าเท่ากัน และเมื่อเปรียบระหว่าง s กับ next ด้วยจำนวนตัวอักษรเท่ากับที่มีอยู่ใน next แล้วยังคงพบว่าเท่ากันอีก แสดงว่ายังมีคำที่มีความยาวมากกว่านี้ ดังนั้นจะสามารถแบ่งคำที่ตำแหน่งนี้ได้ หรืออาจจะรอสะสมตัวอักษรให้มากกว่านี้แล้วไปแบ่งในตำแหน่งต่อไปก็ได้เช่นกัน

4. สถานะที่บังคับให้ต้องแบ่ง

ถ้าเปรียบเทียบตัวอักษรที่อยู่ใน s ด้วยจำนวนตัวอักษรเท่ากับที่มีอยู่ใน current แล้วพบว่าเท่ากัน และเมื่อเปรียบระหว่าง s กับ next ด้วยจำนวนตัวอักษรเท่ากับที่มีอยู่ใน next แล้วพบว่าไม่เท่ากัน แสดงว่าคำที่ยาวที่สุดที่มีคือคำใน current ดังนั้นจะต้องแบ่งคำตรงตำแหน่งนี้ จากนั้นจึงค่อยรับตัวอักษรเข้าไปเริ่มต้นคำใหม่

3.2.3 อัลกอริธึม Backtracking Word Separation (BWS Algorithm)

จากการได้สถานะทั้งสี่ข้างต้นนี้ สามารถนำมาพัฒนาให้เป็นอัลกอริธึมสำหรับตัดคำได้อย่างสมบูรณ์ที่เรียกว่า Backtracking Word Separation หรือ BWS

เนื่องจากสถานะทั้งสี่นั้นสามารถเขียนแทนได้ด้วยทรัยหรือ NFA ซึ่งก็คือรูปแบบของทรีชนิดหนึ่ง (separation tree) และจุดแบ่งคำที่ได้จากแต่ละสถานะก็ได้ มาจากตำแหน่งของโหนดที่อยู่ในทรัยนั้น การค้นหาจุดแบ่งคำหรือสถานะใดๆที่ต้องการก็คือการสร้าง อัลกอริธึมที่เดินเยี่ยมโหนดต่างๆนั่นเอง

อัลกอริธึมนี้จะทำการอ่านตัวอักษรจากสตริงของอักขระที่ป้อนเข้ามา ทีละตัว จากนั้นก็จะเดินไปตามเส้นทางที่อยู่ในทรัยเพื่อค้นหาสถานะต่างๆ ซึ่งถ้าใช้วิธีการเขียนโปรแกรมในลักษณะของการเรียกตัวเอง (recursive) จะเขียนได้สะดวกและง่ายที่สุด แต่วิธีนี้จะทำงานได้ค่อนข้างช้าเพราะเมื่อถึงจุดที่แบ่งได้นั้น อัลกอริธึมจะมีทางเลือก (candidate) ที่จะเดินได้สองทางคือการจะแบ่งหรือไม่แบ่ง ในขั้นนี้อัลกอริธึมจะลองเลือกทางใดทางหนึ่ง ซึ่งถ้าทำแล้วสำเร็จก็จะได้จุดแบ่งคำตามต้องการ ในทางตรงกันข้ามถ้าเลือกทางนั้นแล้ว

ไม่สำเร็จก็จะต้องย้อนกลับมาอีกทางหนึ่งทำให้เกิดการเสียเวลาขึ้น ถ้าจำนวนคำมีมากขึ้นเวลาที่จะต้องเสียไปนี้จะเพิ่มมากขึ้นด้วยเช่นกัน

ในการออกแบบอัลกอริธึมตัดคำของโปรแกรม CU Writer แก้ปัญหาการเสียเวลาในการหาจุดแบ่งคำนี้โดยใช้วิธีการเขียนโปรแกรมแบบไม่เรียกตัวเอง (non-recursive) โดยจะแบ่งหน้าที่ของอัลกอริธึมออกเป็นสองส่วน ส่วนแรกจะทำหน้าที่สแกนสตริงและพิจารณาสถานะของทรัยที่กำลังเดินไปจนกระทั่งไปต่อไม่ได้หรือจบสตริงนั้นหลังจากนั้น จะเข้าสู่ส่วนที่สองซึ่งจะทำหน้าที่แบ็กแทรก (back track) ในส่วนนี้จะทำหน้าที่ตรวจสอบหาคำที่ยาวที่สุดที่สามารถเป็นไปได้ เมื่อผ่านขั้นตอนนี้แล้วก็จะได้จุดแบ่งคำตามต้องการ