# ระบบตรวจสอบความเหมือนกันของตัวโปรแกรม

PLAGIARISM DETECTION SYSTEM

คีริน ก๊กเครือ

แทน แก้วร่วมวงศ์



ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2550

ปริญญานิพนธ์ปีการศึกษา 2550

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ระบบตรวจสอบความเหมือนกันของตัวโปรแกรม

PLAGIARISM DETECTION SYSTEM


ผู้จัดทำ

1. นายคีริน ก๊กเครือ รหัสนักศึกษา 47010078
2. นายแทน แก้วร่วมวงศ์ รหัสนักศึกษา 47010293


_____ อาจารย์ที่ปรึกษา

(อาจารย์เกียรติกูล เจียรนัยธนะกิจ)

# ระบบตรวจสอบความเหมือนกันของตัวโปรแกรม

นายคีริน ก๊กเครือ 47010078
นายแทน แก้วร่วมวงศ์ 47010293
อาจารย์เกียรติกูล เจียรนัยธนะกิจ อาจารย์ที่ปรึกษา
ปีการศึกษา 2550

**บทคัดย่อ**

เป็นที่ทราบกันดีอยู่แล้วว่า ในสถานศึกษาหนึ่งๆ นั้น มักมีนักเรียนบางส่วนพยายามที่จะบ่ายเบี่ยงภาระหน้าที่ในการศึกษาของตน ซึ่งการลอก ก็ถือเป็นหนึ่งในวิธีเหล่านั้นด้วย และการตรวจสอบด้วยมนุษย์นั้นสิ้นเปลืองเวลาและทรัพยากรมาก จึงมีการคิดค้นระบบตรวจสอบแบบอัตโนมัติขึ้น

ในปริญญานิพนธ์ฉบับนี้จะนำเสนอเทคนิคการตรวจจับการลอกในงานเขียนโปรแกรมด้วยวิธี พีดีเทคต์ และ เจแพลก รวมไปถึงการนำมาใช้งาน ข้อดีข้อเสีย และการนำทั้งสองเทคนิคนี้มารวมกันเพื่อให้เกิดผลในการตรวจสอบที่ละเอียดที่สุด และผลทดสอบจากการนำโปรแกรมที่ได้ไปใช้งานจริง

# Source Code Plagiarism Detection


| | | |
|---|---|---|
| Mr.Kerin | Kokkhour | 47010078 |
| Mr.Tan | Keawroumwong | 47010293 |
| Mr.Kietikul | Jiaranaithanakit | Advisor |

Academic year 2007



## Abstract

As we know, in academic environment there tends to be some students who try to avert one own duty in educating oneself. And Plagiarism is one of such act. To detecting Plagiarism with human is wasting resources, Tiring and also troublesome. This brings the development of automated plagiarism detection systems.

In this report, we present plagiarism detection technics called "Pdetect" and "Jplag" Including how to implement, strong and weak point and the way to combine those technics to create the most precision results and also the results of the program in actual use.

# สารบัญ

หน้า

# สารบัญตาราง

หน้า

# สารบัญภาพ

หน้า

# บทที่ 1
# บทนำ

## 1.1. ความเป็นมาและความสำคัญของปัญหา

เราสามารถพบได้บ่อยครั้งว่า มีนักศึกษาหลายคน ที่กระทำการไม่ซื่อในการส่งงาน การลอกงานกันในชิ้นงาน Programming นั้นก็เป็นหนึ่งในการกระทำดังกล่าว ซึ่งเป็นผลเสียอย่างมากต่อการพัฒนาการศึกษา ดังนั้น การตรวจหาการลอกกันในชิ้นงาน programming นั้นจึงเป็นหนึ่งในสิ่งจำเป็นที่ผู้สอนต้องกระทำ แต่ว่าการทำเช่นนั้นเป็นการสิ้นเปลืองเวลามากถ้าทำด้วยตัวเอง อีกทั้งการตรวจสอบยังยุ่งยากเพราะนักศึกษามักเปลี่ยนแปลงแก้ไขงานไม่ให้ถูกจับได้ ดังนั้นการตรวจจับการลอกแบบอัตโนมัติจึงได้มีการพัฒนามาอย่างต่อเนื่องตั้งแต่อดีตจนถึงปัจจุบัน

## 1.2. วัตถุประสงค์

วัตถุประสงค์ของชิ้นงานนี้คือการสร้างระบบตรวจจับการลอกงานเขียนโปรแกรมแบบอัตโนมัติที่มีประสิทธิภาพ โดยใช้เทคนิคที่มีอยู่แล้วหรือนำมาพัฒนาเพิ่มเติมให้มีประสิทธิภาพมากยิ่งขึ้น

### 1.2.1 ขอบเขตของปัญหา

จะต้องตรวจจับการลอกกันด้วยวิธีที่มีการหลบเลี่ยงต่างๆ ได้อย่างถูกต้อง ต้องทำงานได้รวดเร็วและแม่นยำ ไม่กินทรัพยากรของเครื่อง และใช้งานง่าย แสดงข้อมูลที่เกี่ยวข้องในการวิเคราะห์ได้

### 1.2.2 การศึกษาเพื่อแก้ปัญหา

การศึกษาด้านการตรวจจับ Plagiarism นั้นได้รับการพัฒนามาจากอดีตสู่ปัจจุบัน และมีพัฒนาการมาอย่างต่อเนื่อง โดยวิธีที่ได้มีการคิดค้นขึ้นมานั้นประกอบไปด้วยวิธีต่อไปนี้

### 1.2.3 การดำเนินการเพื่อแก้ไขปัญหา

ทำการศึกษาทฤษฎีที่เกี่ยวข้อง หาวิธีตรวจจับที่มีประสิทธิภาพ แก้ไข ปรับปรุง เปลี่ยนแปลงทฤษฎีที่มีอยู่ให้ได้ประสิทธิภาพสูงสุด พัฒนาโปรแกรม และทดลองใช้ เก็บข้อมูลเพื่อทำการแก้ไขเพิ่มประสิทธิภาพการทำงานต่อไป

# บทที่ 2
# ทฤษฎีเบื้องต้นเกี่ยวกับระบบตรวจสอบความเหมือนกันของตัวโปรแกรม

## 2.1. Attribute Counting Method

วิธีการใช้ Attribute counting นี้ เป็นวิธีการนับจำนวนของ attribute ที่มีอยู่ในตัวโปรแกรมว่า มีชนิดละเท่าไหร่ซึ่งสิ่งที่จะนำมานับเป็น attribute ตามที่ได้ศึกษามามีอยู่ 6 วิธี ดังนี้

**2.1.1 Volume** เป็นวิธีการนับจำนวนของตัวแปร และเครื่องหมาย ที่มีอยู่ภายในตัวโปรแกรม โดยจะมีสิ่งที่สนใจอยู่ 4 อย่างคือ

- n1 จำนวนของเครื่องหมายที่มีอยู่ในตัวโปรแกรม
- n2 จำนวนของตัวแปรที่มีอยู่ในตัวโปรแกรม
- N1 จำนวนการใช้เครื่องหมายทั้งหมดในตัวโปรแกรม
- N2 จำนวนการใช้ตัวแปรทั้งหมดในตัวโปรแกรม

ซึ่งค่า Volume นั้นสามารถคำนวนได้จาก สมการ $V= (N1 + N2 ) \log2 (n1+n2)$

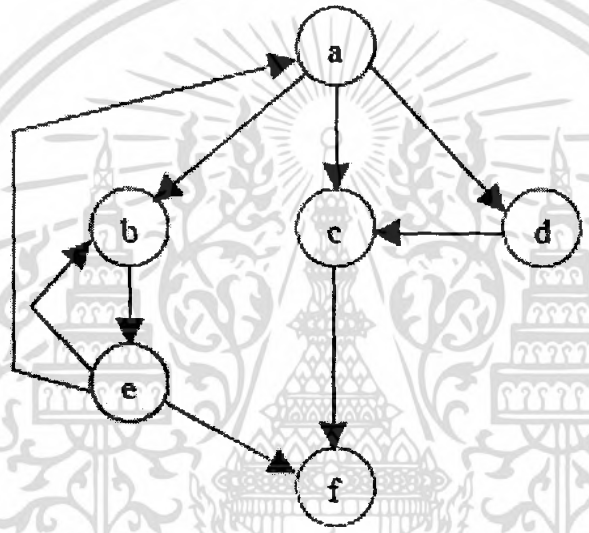
**2.1.2 Structure** วิธีการนี้ จะพิจารณาถึงลักษณะของโปรแกรมว่ามีกี่ขั้นตอนแล้วนำมาแสดงผลเป็นแผนผังการส่งต่อข้อมูล

**2.1.3 Data Dependency** จะเป็นการวัดด้วยการคาดเดาสาเหตุของการใช้ data หรือการประกาศตัวแปรแล้วนำมาแสดงผลในรูปของ node และ flow graph

**2.1.4 Nesting Depth** วิธีนี้ จะใช้การกำหนดความลึกให้กับ code แต่ละบรรทัด แล้วนำมาหารด้วย statement ที่มีทั้งหมดในตัวโปรแกรม เพื่อให้ได้ออกมาเป็น ค่าความลึกเฉลี่ยของตัวโปรแกรม

**2.1.5 Control Structure** เป็นวิธีการที่จะกำหนดค่า weight ให้กับ control structure ต่างๆ เช่น if ... then กำหนด weight 5 เป็นต้น หลังจากนั้น จะนำค่า weight ของ control structure ทั้งโปรแกรมมารวมกันจะได้เป็น program complexity

**2.1.6 Control Flow** วิธีการนี้จะใช้วิธีการวัดโดย ใช้ McCabe's Cyclomatic Complexity ซึ่งจะ ทำงานโดยขั้นแรก จะนำ code program มาทำเป็น list แล้วแปลงให้เป็น control graph ที่มีทางเข้า และทางออกของ node ที่เฉพาะเจาะจง ซึ่งวิธีนี้ค่อนข้างจะเป็น Language Dependent กล่าวคือ จะต้องเข้าใจการทำงานของ ตัว syntax และ semantics ของ code นั้นเสียก่อน จึงจะสามารถนำมาสร้างเป็น control graph ได้ ซึ่งในแต่ละ node จะแสดงถึง block ของ code และ edge จะแสดงถึง sequential ของตัวโปรแกรม




**ภาพที่ 2.1 : กราฟที่ใช้วิธีแมคเคบส์**

ในการสร้างกราฟนั้น จะมีข้อบังคับอยู่ 2 อย่างคือ

1. จะต้องสามารถ เข้าถึงทุก node ได้ หากว่าเริ่มที่ start node
2. node อื่นๆ ทุก node จะต้องสามารถเดินทางไปยัง node finish ได้

หลังจากสร้างกราฟได้แล้ว วิธีการโดยภาพรวมที่จะใช้ในการตรวจหา Complexity ของ program หรือการคำนวนหาค่า Linearly independent path v(G) (โดยที่ G จะหมายถึง graph ที่ได้สร้างขึ้นมา)

จะมีคุณสมบัติอยู่จำนวนหนึ่งที่ต้องคำนึงถึง ดังนี้

1. $v(G) \geq 1$
2. v(G) เป็นเป็นตัวเลขที่มากที่สุดของ linearly independent path ใน G
3. การใส่เพิ่ม หรือการตัด funtional statement ออกใน G จะไม่ส่งผลกับ v(G)
4. ถ้า v(G) = 1 จะทำให้ G นั้น มีเพียง path เดียว
5. การเพิ่ม Edge ใน G จะส่งผลให้ต่อ v(G)
6. v(G) จะขึ้นอยู่กับค่า structure ของ G เท่านั้น

จากคุณสมบัติที่ได้เกล่าวมาเบื้องต้นแล้ว เราสามารถพิจารณาหาค่า v(G) ได้จาก สมการ v(G) = e - n + 2p
ซึ่ง

e คือจำนวนของ edge ในกราฟ
n คือจำนวนของ node ในกราฟ
p คือจำนวนของ connected component *ยกตัวอย่างเช่น p=1 นั่นคือมีเพียง module เพียง , p=2 เมื่อพิจารณา module สองตัว โดยที่ $p \neq 0$.

อันที่จริงแล้ววิธีการข้างต้นนั้นไม่ได้พัฒนามาเพื่อการตรวจหา plagiarism แต่ก็ยังสามารถใช้ได้เนื่องจาก code ที่มีความคล้ายคลึงกันของค่า complexity จะสามารถเรียกได้ว่า มีความ น่าสงสัยว่าจะเป็นการลอกกันของ code และ จะนำไปตรวจด้วยตาของมนุษย์ ต่อไป จากวิธีการตรวจสอบแบบ McCabe states เป็นที่น่าสนใจอย่างหนึ่งว่า สไตล์การเขียน ของแต่ละ programmer นั้นจะมีความสัมพันธกับค่า complexity measure ของโปรแกรมด้วย

## 2.2. Structure Metric Method

เป็นวิธีการตรวจสอบ ที่เน้นใช้ structure ของตัวโปรแกรมเป็นหลัก ในการตรวจสอบ ตามที่ได้ศึกษามามี 3 วิธี ดังนี้

**2.2.1 Dotplot** ด้วยเทคนิคนี้จะเป็นการแสดง pattern ของ string m match กันระหว่าง code สองอัน (หรือ text ใดๆ ) ออกมาเป็นภาพให้เห็น ซึ่งวิธีการ dotplot นั้นถึงแม้จะไม่เป็นวิธีการที่เกี่ยวข้องกับภาษา programing โดยตรงอาศัยความเข้าใจในภาษา ทั้ง syntax และ semantics ของ code ที่จะนำมาเปรียบเทียบก่อน ซึ่งนั่นจะเป็นจุดที่ทำให้วิธีนี้มีความยืดหยุ่นในการ ตรวจสอบสูงมาก ซึ่งข้อได้เปรียบของวิธี dotplot นั้นก็คือการใช้หลักการของการมองเห็น ของมนุษย์เข้ามาช่วย

ในการตรวจสอบความเหมือน ดังที่ Helfman กล่าวไว้ว่า วิธีก่อนๆ ที่ใช้ในการตรวจสอบความเหมือน เช่น algorithms ที่หา substring ที่ยาวที่สุด นั้นจะไม่แสดง ให้เห็นถึงความเหมือนของตัว structure ที่ซ่อนอยู่ในตัวโปรแกรม , ข้อมูล , และภาษา แต่อย่างไรก็ตาม การใช้ หลักการมองเห็นของมนุษย์เข้ามาช่วยนั้น เป็นข้อด้อยสำคัญอีกเช่นกัน

เนื่องจาก ว่า การตัดสินว่า อันไหน เป็น code ที่มีการ copy มาหรือ อันไหนไม่ได้ถูก copy มานั้น จะทำได้ยากด้วย ซึ่งก็หมายความว่า เราจะไม่สามารถ ประเมินค่าความเหมือนออกมา เป็นค่าเลข หรือค่า percent ได้โดยง่าย ซึ่ง จากที่กล่าวมาแล้ว ก็จะมาดูวิธีการในการพิจารณา code , โดย นำตัวโปรแกรม มาแปลงเป็น sequence ของ token แล้วนำทั้งสองชุด มารวมกัน

ยกตัวอย่างโปรแกรมที่มีแต่ส่วน comment เช่น

โปรแกรมชุดแรก : //test this

โปรแกรมชุดสอง : //we must test this now

would be tokenised into the final sequence

จะมีการ tokenize ให้เป็น sequence ของ string ดังนี้

test this we must test this now

หลังจากได้ sequence แล้วเราก็นำไปสร้างเป็นตาราง ดังนี้

**ตารางที่ 2.1 แสดงตัวอย่างการ Tokenize โค้ด**

| | test | this | we | must | test | this | now |
|---|---|---|---|---|---|---|---|
| test | • | | | | • | | |
| this | | • | | | | • | |
| we | | | • | | | | |
| must | | | | • | | | |
| test | • | | | | • | | |
| this | | • | | | | • | |
| now | | | | | | | • |

หลังจากที่ นำมา plot เป็นตารางแล้ว จุดที่อยู่นอกเหนือจากตรงกลางจะแสดงถึง subroutine ที่ถูก copy มา ซึ่งจากตรงนี้ จะแสดงให้เห็นว่าวิธี dotplot นั้น จะสามารถ ทนต่อการที่ copy subroutine มาแล้วนำมาเปลี่ยนแปลงลำดับของโปรแกรมใหม่

**2.2.2 Yap3** เป็นวิธีที่ใช้หลักการของ greedy string tiling เข้ามาช่วยในการตรวจสอบ โดยการทำงานจะแบ่งออกเป็น 2 ขั้นตอนคือ

ในขั้นตอนแรก นั้นจะทำการแปลง code ทั้งสองตัวให้อยู่ในรูปของ token ของ string ซึ่ง มีลำดับดังต่อไปนี้

- ตัด comment และ string constant ออก
- แปลง อักษรตัวใหญ่ให้เป็นตัวเล็ก
- เปลี่ยนสิ่งที่คล้ายๆ กันให้อยู่ในรูปทั่วไป เช่น เปลี่ยน ฟังก์ชั่นให้เป็น procedure
- จัดเรียงตำแหน่งของฟังก์ชั่นใหม่ตามลำดับการcallแล้วนำไปใส่ในตัว program ตามตำแหน่งที่call
- ตัด token ที่ไม่ได้เป็น reserved word หรือ built in functon ของภาษานั้นออก

ต่อมาคือขั้นตอนในการเปรียบเทียบ ซึ่งจะใช้ greedy string tiling เข้ามาช่วย ซึ่งมีขั้นตอนดังนี้

- เป็นการจับคู่แบบ one to one เพื่อหา substring ระหว่าง source file และ target file
- เมื่อ tokenใดถูกนำไปเป็นsubstring แล้วจะถูก mark ว่า ตัวนี้ใช้ไปแล้ว เพื่อป้องกันการนำไปใช้ซ้ำ
- จะได้ค่า Maximal Match มาเพื่อบอกว่า มีจำนวน token ที่เหมือนกันระหว่าง source และ target ไฟล์เท่าไหร่
- มีการใส่ค่า Mininal Match Length เข้าไป เพื่อจะได้ไม่ต้องสนใจ substring บางอันที่สั้นเกินไป เพื่อเพิ่มความแม่นยำในการตรวจสอบ

**2.2.3 Plague** วิธีนี้มีการทำงานที่คล้ายกับ Yap3 แต่จะมีการทำงาน แบบ 3 ขั้นตอนแทนคือ

- ขั้นตอนที่ 1 สร้าง sequence ของ token และ list ของ structure metric จาก structure profile ซึ่งตัว profile นั้นจะรวบรวมสรุป control structure ที่ใช้ใน program และนำเสนอ iteration และ selection และ statement block
- ขั้นตอนที่ 2 ใช้ O(n2) phase เปรียบเทียบ structure profile และแยกแยะ pair ที่ใกล้เคียงกัน
- ขั้นตอนที่ 3 เปรียบเทียบtoken sequence โดยใช้longest common subsequence เพื่อหาค่าSimilarity

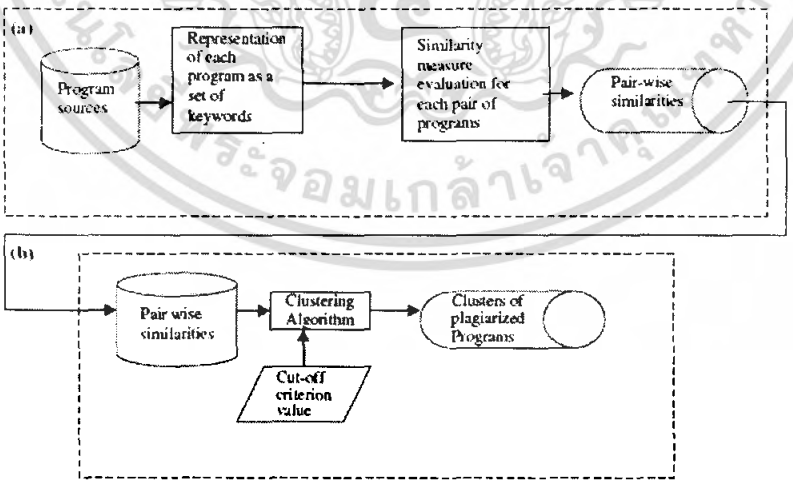
# บทที่ 3
# อัลกอริธึมที่ใช้พัฒนาโปรแกรมตรวจจับ

หนึ่งในระบบที่ถูกพัฒนาขึ้นนั้นก็คือ Pdetect ซึ่ง Pdetect ทำงานด้วยวิธี Attribute counting ซึ่งจะมองโปรแกรมหนึ่งๆ เป็นชุดรูปแบบของ Token ซึ่งแปลงมาจาก Operator และ Operand โดยโปรแกรมเหล่านี้จะถูก Normalize มาเป็นชุดของ Token และจะนำมาเปรียบเทียบว่ามี Token ชนิดใดบ้างที่ใช้ เหมือนกันกี่ที่ จำนวนเท่าไร และให้ผลออกมาเป็นความเหมือนกันของคู่โปรแกรมในหน่วยเปอร์เซ็นต์ ซึ่งระบบ Pdetect ก็มีความแม่นยำในการตรวจจับสูง

แต่เนื่องจาก PDetect ใช้การมอง Keyword ของชนิด Operand และ Operator จึงมีความบกพร่องถ้าโปรแกรมที่ทำการ Plagiarism แบบเปลี่ยนชนิดของตัวแปร เพราะจะทำให้ PDetect มองเห็นเป็นคนละรูปแบบ Token ซึ่งในเอกสารนี้จะกล่าวถึงการนำอีกวิธีตรวจจับเข้ามาเสริมเพื่อลบจุดอ่อนนี้ของ PDetect ออกไป วิธีการดังกล่าวก็คือ Jplag นั่นเอง

## 3.1. Pdetect

Pdetect ประกอบไปด้วยส่วนของการแสดง Source code, ส่วนการตัดสินความคล้ายคลึงกัน (Similiarity) และส่วนของการจัดกลุ่ม (Clustering) ซึ่งจะทำการตรวจจับกลุ่มของชิ้นงานที่เป็นไปได้ว่าจะ Plagiarism โดยวิธีของส่วนต่างๆ จะทำการอธิบายต่อไป




**ภาพที่ 3.1 : ระบบโดยทั่วไปของ Pdetect**

### 3.1.1 เค้าโครงโดยสรุป

Pdetect แบ่งได้เป็นสองช่วงโดยช่วงแรกจะทำการประมวลผลแยกคู่ของ Program ที่คล้ายกันจากชุดของโปรแกรมที่ได้มา และช่วงที่สอง ค่าความคล้ายคลึงนี้จะถูกนำไปเข้า Algorithm สำหรับจัดกลุ่มของ Plagiarism ที่ตรวจพบ

ในส่วนแรก (รูปที่ 1) จะรับ Input เป็นกลุ่มของโปรแกรม และแต่ละโปรแกรมในกลุ่มจะถูกนำเสนอในรูปแบบของ Keyword และโปรแกรมเหล่านั้นจะถูกนำมาคำนวณหาค่าความคล้ายคลึงกัน ผลจากส่วนแรกนั้นจะออกมาในรูปแบบของ Text file (Pairwise similiarity) ซึ่งในแต่ละบรรทัดจะมีตัวระบุชื่อของโปรแกรมสองตัวอยู่ และค่าความคล้ายคลึงกัน (Similiarity) ของคู่โปรแกรมนั้น ซึ่งไฟล์นี้จะถูกแสดงให้ผู้ใช้ทราบ เพื่อให้ผู้ใช้ตัดสินว่าคู่โปรแกรมใดบ้างที่เป็น Plagiarism pair ซึ่งโดยทั่วไปแล้ว การที่ผู้ใช้จะนั่งตรวจสอบ Code ทั้งหมดที่มีการหาค่า Similiarity มานั้นเป็นเรื่องที่กินเวลามาก จึงได้มีการกำหนดค่าที่เรียกว่า Bottom Treshold เอาไว้ โดยที่ค่า Similiarity ที่ต่ำกว่า Treshold นี้จะไม่ได้แสดงขึ้นมาให้ผู้ใช้ตัดสินใจ เพราะว่า งานที่ Plagiarize มานั้น มักมีค่า Similiarity สูง ซึ่งค่า Treshold นี้มีชื่อว่า Cutoff Criterion value และหลังจากนั้นค่าเหล่านี้จะถูกส่งไปที่ส่วนที่สองต่อไป

ในระหว่างส่วนที่สอง ทุกคู่ของโปรแกรมซึ่งมีค่าสูงกว่าหรือเท่ากับ Cutoff criterion value จะถูกนำมาแสดงอยู่ในรูปของ Non-directed graph ถ่วงน้ำหนัก ซึ่งมี Vertices แทนตัวโปรแกรมและ Edge แทนค่าความคล้ายคลึงของแต่ละโปรแกรม และกราฟจะถูกนำมาจัดกลุ่ม (Clustered) ด้วยอัลกอริธึมสำหรับจัดกลุ่มที่เหมาะสม

#### 3.1.1.1 การนำเสนอ Source code

จากคำจำกัดความของคำว่า Plagiarism โดย Parker และ Hamblen โปรแกรมที่มีการ Plagiarized คือ *"โปรแกรมที่สร้างจากอีกโปรแกรมหนึ่ง ซึ่งมีการปรับแต่งเล็กน้อย โดยไม่มีความเข้าใจในตัวโปรแกรมจริงๆ"* ซึ่งถือว่าถูกต้อง เพราะนักศึกษาที่มีความรู้นั้น มักไม่ทำการ Plagiarize และนอกจากนั้น ถ้าสามารถแก้ไขไปจนถึงการเปลี่ยนแปลงโครงสร้าง โดยที่มีความเข้าใจในตัว Code อย่างถ่องแท้ ดังนั้น จากมุมมองของผู้สอน การ Plagiarize นี้อาจไม่ได้ไม่เป็นที่ต้องการเสียทีเดียว

อย่างไรก็ตาม ความยากในการตรวจจับ Plagiarism นั้น มาจากการที่นักศึกษาพยายามดัดแปลงในหลายๆ รูปแบบเพื่อหลีกเลี่ยงการตรวจจับ โดยรูปแบบในการดัดแปลงเหล่านี้ หรือเรียกอีกอย่างว่า Types of attack มีดังต่อไปนี้

- แก้ไข Format โดยการเปลี่ยน Line breaks, spaces, TAB
- เพิ่ม แก้ไข หรือลบ Comment
- แก้ Output ของโปรแกรมหรือ Format ของ Output
- เปลี่ยนชื่อตัวแปร, Method หรือ Class

- รวมและแยกการประกาศตัวแปร
- แก้ Modifier
- ดัดแปลงค่า Constant
- การส่งงานที่ Copy มาตรงๆ โดยหวังว่าจะตรวจจับไม่พบ
- จัดเรียงรูปแบบและตำแหน่งการวางใน Block ของการประกาศตัวแปร
- จัดตำแหน่งการวางของการประกาศ Variable และ Method ทั้งหมด
- การแก้ไข Control Structure เช่น แก้ For loop เป็น While loop, เปลี่ยน loop ธรรมดาเป็น Infinite loop ที่มี Break, เปลี่ยน Switch ไปใช้ If แทน, ใส่ Break statement ที่ไม่จำเป็นในทุกๆ Case ของ Switch เป็นต้น
- การสร้างตัวแปรและ Subexpression เพิ่ม เช่น ย้าย Subexpression ไปเป็น Variable หรือ กลับกัน, ย้ายส่วน Initialize ออกจากส่วนประกาศ เป็นต้น
- แก้บางส่วนของ Method ที่มีอยู่แล้ว ให้เป็น Method ใหม่
- แก้ไขดัดแปลง Scope
- จัดเรียง Independent statement ใน Block เดียวกันใหม่
- แก้ Mathematical Identities.
- แก้ Data structure บางส่วน
- Redesign Structure ใหม่ทั้งหมด

เราสามารถสังเกตุได้ว่า ไม่มีการโจมตีแบบใดเลยที่มีการเปลี่ยนแปลงใน Set ของ Keyword อย่างเห็นได้ชัด ดังนั้น ในการตรวจสอบ เราจึงควรใช้ Index ของ keyword แต่ว่าการใช้ก็ยังมีปัญหาอยู่ คือ Keyword ธรรมดาไม่สามารถครอบคลุมทั้งหมดของโปรแกรมเพื่อความสามารถในการตรวจจับหา Plagiarism ที่ดีได้ จึงมีการใช้เป็น Index ของ Substitute Keyword แทน

**ตารางที่ 3.1 แสดงตัวอย่างโค้ดที่นำมาใช้กับอัลกอริธึม Pdetect**

```
void init(int& start, int& end) {
startGame=false;
for (int i=0; i<4; i++) players[i]=0;
string message="Snakes and Ladders";
setTitle(message);
start=1;
end=100;
back=0;
}
```

```
void initGame(int& first, int& last) {
int i=0;
while (i<4) { person[i]=0; i++; }
string gameTitle="Snakes and Ladders";
title(gameTitle);
playOn=false;
positions=0;
first=1;
last=100;}
```

**ตารางที่ 3.2 แสดง Keyword ที่ได้จากโปรแกรมตัวอย่าง**

*Init={void1, int1, int2, false1, for1, int3}*
*InitGame={void1, int1, int2, int3, while1, false1}*
a: Using only language keywords. Common keywords 4. Total keywords used 6.
*Init={void1, &1, &2,int1, int2, false1, for1, int3, [1,string1}*
*InitGame={void1, &1, &2,int1, int2, int3, while1, [1,string1, false1}*
b: Using user-defined keywords. Common keywords 9. Total keywords used 11.
*Init={void1, &1, &2,int1, int2, false1, loop1, int3, [1,string1}*
*InitGame={void1, &1, &2,int1, int2, int3, loop1, [1,string1, false1}*
c: Using substitute keywords. Common keywords 10. Total keywords used 10.
*Init={void1, &1, &2,&3,&4,int1, int2, false1, loop1, int3, [1,[2,string1}*
*InitGame={void1, &1, &2,&3,&4,int1, int2, int3, loop1, [1,[2,string1, false1}*
d: Using weighted substitute keywords. Common keywords 13. Total keywords used 13.

จากรูปเราสามารถสังเกตได้ว่า โปรแกรมสองตัวนี้มีจุดที่คล้ายคลึงกันอยู่มาก กล่าวคือ ทั้งคู่ Return ค่า void, รับค่า Integer สองค่าเป็น Parameters, ประกาศ String, สร้าง Boolean false, Integer 1, 100 และ 0, และท้ายสุดคือสร้าง 4-position integer array โดยใช้ Loop และในตารางที่ 2 คือ Keyword ที่พบ จะเห็นได้ว่าการดูโปรแกรมด้วย Keyword พื้นฐานนั้นยังไม่พอ จึงควรกำหนดให้ผู้ใช้สามารถเลือกใส่ Keyword เพิ่มได้เองตามที่เห็นสมควร

**ตารางที่ 3.3 แสดง Keyword ที่ใช้กับโปรแกรมตัวอย่าง**

| |
|---|
| (void,int,false,for,while,&,[,string) |
| a. User defined keywords |
| (void,int,false,(for,loop),(while,loop),&,[,string) |
| b. Substitute keywords |
| (void,int,false,(for,loop),(while,loop),(&,2),([,2),string) |
| c. Weighted substitute keywords |

ในรูปแบบ B นั้น มีการใช้ Substitute keyword เพื่อมอง for และ while เป็น Loop เหมือนกัน ก็จะสามารถตรวจจับการเปลี่ยนแปลงรูปแบบของ Loop ได้ ซึ่งรูปแบบการตรวจจับนั้นขึ้นอยู่กับ Keyword ที่ผู้ใช้ Define เพิ่มขึ้นมาด้วย ส่วนในรูปแบบ C จะมีการถ่วงน้ำหนักให้บาง keyword เพื่อเน้นความสำคัญของมันตอนที่ตรวจสอบ หากผู้ใช้คิดว่าส่วนนี้น่าจะมีการลอกกันได้ง่าย

ซึ่งจากรูปแบบการตรวจสอบเช่นนี้นั้น เราสามารถที่จะใช้การตรวจสอบแบบ Pdetect กับโปรแกรมชนิดอื่นได้ ไม่จำเป็นว่าต้องเฉพาะ C++ เพราะเราสามารถเปลี่ยนแปลง Keyword ให้เหมาะสมได้

หลังจากเสร็จสิ้นแล้ว เราจะได้ Output เป็น Text file (Pairwise similiarity) ซึ่งในแต่ละบรรทัดจะมีตัวระบุชื่อของโปรแกรมสองตัวอยู่ และค่าความคล้ายคลึงกัน (Similiarity) ของคู่โปรแกรมนั้น โดยจะนำมาเป็น Input ของส่วนที่สอง ก็คือการทำ Clustering ซึ่งเราจะสร้างกราฟถ่วงน้ำหนักแบบ Non-directed ซึ่งจะใช้ค่าที่มาถ่วงน้ำหนักเป็นค่า Similiarity ของคู่โปรแกรมที่มากกว่าค่า Cutoff Criterion Value ที่ตั้งไว้ ซึ่งทำให้การตรวจจับกลุ่มของ Plagiarism กลายเป็นการทำ Clustering ของ Graph ที่สร้างขึ้น ซึ่งเราจะใช้ Algorithm ที่ชื่อ WmajorClust ซึ่งทำหน้าที่ Graph-clustering ที่มีพื้นฐานมาจากการเพิ่มประสิทธิภาพของ Connectivity ให้สูงสุด โดยมี Algorithm ดังนี้

```
void wMajorClust(Graph G) {
 int n=0; boolean t=false;
 for each vertex V in G V.clusterId= n++;
 while (!t) {
 t=true;
 for each vertex V in G {
  int clusterNo=maxNeibCluster(V);
  if (V.clusterId!=clusterNo) {
  V.clusterId=clusterNo;
  t=false;
 } } } }
```

```
int maxNeibCluster(Vertex v) {
 float weights[];
 for each edge E connected to V {
  Vertex neib=null;
  if (V.vertexId=E.vertexId1) neib=Vertex(E.vertexId2);
  else neib=Vertex(E.vertexId1);
  weights[neib.clusterId]+ =E.weight;
 }
 return maxPos(weights);
 }
```

**ตารางที่ 3.4 แสดงอัลกอริธึมของ WmajorClust**

WmajorClust จะจัดค่าให้ Cluster ในแต่ละ Node ของ Graph และนำไปสู่การรวม Node เข้ากับ Cluster โดยวัดจากน้ำหนักของ Edge ซึ่งผลลัพท์ที่ได้จะแสดงกลุ่มของโปรแกรมที่น่าจะ Plagiarize กัน เป็นกลุ่มๆ ออกมาแสดงให้แก่ผู้ใช้ เป็นอันเสร็จสิ้นกระบวนการตรวจสอบ

### 3.1.2 จุดอ่อนและจุดแข็ง

จากการทดสอบโดย L. Moussiades และ A. Vakali [2] แสดงให้เห็นว่า Pdetect มีจุดแข็งและจุดอ่อนอยู่ ซึ่งจากการทดลองให้ทำการประมวลผลกับโปรแกรม ทำให้เราทราบว่า Pdetect นั้น มีประสิทธิภาพสูงในการตรวจจับ Plagiarism ในหลายๆ แบบ แต่ยังไม่สามารถตรวจสอบ Plagiarism ประเภทที่มีการแก้ไข Keyword ได้อย่างมีประสิทธิภาพ เนื่องด้วย Pdetect ใช้การตรวจสอบตาม Keyword ที่ตรงกัน ถ้าโปรแกรมมีการแก้ไข Keyword ก็จะทำให้ Pdetect ถือว่าเป็นโปรแกรมคนละชนิดกัน แต่ในด้านการตรวจสอบเทคนิคแบบอื่นๆ เช่นการสลับที่ของ Code นั้น ทำได้อย่างมีประสิทธิภาพ

3.1.2.1 รูปแบบการพัฒนาใหม่

เนื่องจากการที่ Pdetect ยังมีจุดอ่อน ทำให้เรายังไม่สามารถเชื่อถือเฉพาะ Pdetect อย่างเดียวได้ จึงได้มีการทดลองนำวิธีอื่นมาผนวกด้วย ซึ่งวิธีที่ผู้ศึกษาได้หยิบยกขึ้นมาใช้งานนั้นก็คือ Jplag

## 3.2 JPlag

JPlag คือระบบที่เขียนโดยภาษา Java เพื่อใช้ในการตรวจสอบ source code program ในภาษา Java , C , C++ ว่ามีความเป็นไปได้ว่า จะเป็น program ที่ลอกกันมาหรือไม่ ซึ่ง สามารถทดลองใช้ได้ที่ www.jplag.de ซึ่ง JPlag ได้ถูกนำไปใช้อย่างแพร่หลาย ในการตรวจสอบ source code ของหลายๆ ระดับการศึกษา โดย จะรับ Input เป็น ชุดของ source code program มาเป็นชุดๆ แล้วเปรียบเทียบ ความเหมือน ทีละคู่ แล้วแสดงผลออกมา ผ่านหน้า webpage

### 3.2.1 เค้าโครงโดยสรุป

ดังที่กล่าวมาแล้วว่า JPlag จะรับ Input เป็นชุดของ source code program แล้วนำมาเปรียบเทียบ ซึ่งขั้นตอนในการเปรียบเทียบความเหมือน นั้น จะแบ่งเป็น 2 ขั้นตอนหลัก เรียกว่า 2 เฟส โดยแต่ละเฟส มีหน้าที่ดังนี้

เฟส 1 แปลง source code ของ program ที่รับเข้ามาให้เป็น string token

เฟส 2 นำ token ของ string มาเปรียบเทียบความเหมือนกันทีละคู่ แล้วแสดงผลออกหน้า webpage

ซึ่งในส่วนต่อไปจะกล่าวถึงขั้นตอนการทำงานของแต่ละเฟสให้ละเอียดมากขึ้น

3.2.1.1 ขั้นที่ 1 : ขั้นตอนการแปลง Souce code ของ ให้เป็น token string

ในการแปลง source code ให้เป็น token string นั้น จะต้องแปลงโดยให้ token แต่ละตัว แสดงถึงส่วนที่สำคัญของ program เช่น structure , flow control , structure control ซึ่ง ส่วนต่างๆ เหล่านี้ จะเป็นส่วนที่เปลี่ยนแปลงได้ยาก เวลาที่ มีการลอกกัน ซึ่งจะผิดกับ ส่วนที่สามารถเปลี่ยนแปลง ได้ง่าย ที่เป็นรูปแบบผิวเผินเช่น ชื่อตัวแปร ชนิดของตัวแปร ชื่อ ฟังก์ชั่น เป็นต้น ในขึ้นตอนการแปลงนั้น จะไม่สนใจ white space , ชื่อตัวแปร , comment ฯลฯ และจะมีการใส่ข้อมูลเฉพาะเข้าไปในบาง token เพื่อไม่ให้ ตรวจสอบผิดพลาด ในส่วนที่จะต้อง มีอยู่ทุก program เช่น header หรือ ชุดคำสั่งเฉพาะ

3.2.1.2 ขั้นที่ 2 : ขั้นตอนการเปรียบเทียบความเหมือนกันของ string token แต่ละคู่

ในการเปรียบเทียบความหมือนของ source code program ที่ถูกแปลงเป็น token แล้วนั้น จะใช้วิธีการของ Greedy String Tiling ซึ่งเป็นวิธีการ ตรวจสอบ ผ่าน string token เพื่อค้นหา substring ที่ ยาวที่สุดเท่าที่เป็นไปได้ ที่ปรากฏอยู่ ในชุด program ทั้ง 2 ชุด โดย จะไม่มีการ นำ token ที่ถูกนับรวมใน substing ใดๆ มาใช้ ร่วมกับ substring อื่นอีก และเพื่อป้องกัน การตรวจสอบ ผิดพลาดในส่วนของชุดคำสั่งที่เป็นชุดคำสั่ง เฉพาะ หรือ ในส่วนของ header ไฟล์ จึงมีการใส่ค่า Minimun Match Length เข้าไป เพื่อที่เวลาแสดงผล จะไม่ต้องแสดงผล ในส่วนที่เป็น substring ที่มีความยาวต่ำ กว่า ค่า Minimum Match Length ในขึ้นตอนแสดงผล เมื่อ ครบการค้นหา ความเหมือนของชุดโปรแกรม แล้ว ทางระบบ ของ JPlag ก็จะแสดงผลออกมาทางหน้า webpage ว่า ชุดโปรแกรม คู่นั้น เหมือนกันกี่ % และมีบรรทัดใด ถึง บรรทัดใด บ้างที่เหมือนกัน และมีกี่ชุด

3.2.1.3 ตัวอย่างการทำ Tokenize code

จากที่เราได้กล่าวมาแล้วว่า ในเฟสแรก จะมีการแปลง source code program ให้อยู่ในรูปของ string token ดังนั้น ในส่วนนี้ จึงแสดงตัวอย่างการทำ tokenize โดย ตัวอย่างจะเป็น code ในภาษา Java

| | Java source code | Generated tokens |
|---|---|---|
| 1 | public class Count { | Begin Class |
| 2 | public static void main(String[] args) | Var Def, Begin Method |
| 3 | throws java.io.IOException { | |
| 4 | int count = 0; | Var Def, Assign |
| 5 | | |
| 6 | while (System.in.read() != -1) | Apply, Begin While |
| 7 | count++; | Assign, End While |
| 8 | System.out.println(count+" chars."); | Apply |
| 9 | } | End Method |
| 10 | } | End Class |

3.2.1.4 Greedy String Tiling

ขั้นตอน การทำ Greedy String Tiling นั้น จะประกอบไปด้วย nested loop 3 ชั้น คือ

ชั้นที่ 1 ใช้เป็น Iterator ชี้ token ในชุดแรก ไปทีละตัว จนครบทุกตัว

ชั้นที่ 2 ใช้เปรียบเทียบ token ของชุดแรก กับ token ของอีกชุดหนึ่ง จนครบทุกตัว

ชั้นที่ 3 ใช้ ในกรณีที่ หากมีการค้นพบว่า token ของ ชุดแรก ชุดที่ 2 เหมือนกัน loop นี้ จะหาว่า substing นั้นมีความยาวเท่าไร

ซึ่งตัวอย่าง source code ของ algorithm ที่ใช้ในการทำ Greedy String Tiling จะแสดงในหน้าถัดไป

```
0   Greedy-String-Tiling(String A, String B) {
1       tiles = {};
2       do {
3           maxmatch = M;
4           matches = {};
5           Forall unmarked tokens Aa in A {
6               Forall unmarked tokens Bb in B {
7                   j= 0;
8                   while (Aa+j == Bb+j &&
9                       unmarked(Aa+j) && unmarked(Bb+j))
10                      j ++;
11                  if (j == maxmatch)
12                      matches = matches ⊕ match(a, b, j);
13                  else if (j >maxmatch) {
14                      matches = {match(a, b, j)};
15                      maxmatch = j;
16                  }
17              }
18          }
19          Forall match(a, b, maxmatch) ∈ matches {
20              For j = 0. . . (maxmatch − 1) {
21                  mark(Aa+j);
22                  mark(Bb+j );
23              }
24              tiles = tiles ∪ match(a, b, maxmatch);
25          }
26      } while (maxmatch > M);
27      return tiles;
28  }
```

### 3.2.2 จุดอ่อนและจุดแข็ง

JPlag นั้นสามารถ ทนต่อการเปลี่ยนแปลง source code ของ program แล้วยังสามารถ ให้ผลการตรวจสอบ ได้แม่นยำ อยู่ ดังนี้

- การเปลี่ยนรูปแบบการจัดวาง code เช่น เคาะวรรค, tab
- การใส่ หรือ ตัด ส่วน comment
- การเปลี่ยนแปลงลักษณะ output ของ โปรแกรม
- การเปลี่ยนชื่อ หรือ ชนิดของตัวแปร
- การแยก หรือ รวม การประกาศตัวแปร
- การเปลี่ยนแปลงค่า constant ใน program

ซึ่งจากการเปลี่ยนแปลงเหล่านี้ มักจะไม่ส่งผลต่อ Structure ของ program อีกทั้งในบางส่วน จะเป็นส่วนที่ JPlag ไม่นำมาใช้ในขั้นตอนการทำ tokenize จึงทำให้การเปลี่ยนแปลงเหล่านี้ ไม่ส่งผลให้ การตรวจสอบแบบ JPlag นั้นผิดพลาด แต่อย่างใด

อย่างไรก็ตาม JPlag ก็ยังคงมีจุดบอดในกรณีที่มีการเปลี่ยนแปลง code ที่จะส่งผลต่อ structure ของ program ดังนี้

- เปลี่ยนแปลงตัว control structure เช่น ใช้ loop ไม่สิ้นสุด
  ควบคู่กับการ Break ตาม condition แทนการใช้ loop ธรรมดา
- มีการใช้ ตัวแปร แบบ Temporary หรือ การใช้ dynamic allocate
- มีการจัดเรียง, สลับที่ structure ของ program ใหม่
- มีการจัดเรียง statement ที่ stand alone ของ program ใหม่
- มีการใส่ หรือ ตัด ชุดคำสั่งที่ไม่จำเป็น ในตัวโปรแกรม
- มีการ ออกแบบ structure ของโปรแกรมใหม่

ซึ่งจากการเปลี่ยนแปลงในลักษณะนี้เอง ที่เป็นจุดด้อยของ JPlag ที่ไม่สามารถตรวจสอบได้ ดังนั้น ผลที่ได้ จากการตรวจสอบ JPlag ที่มี โปรแกรมที่ถูกดัดแปลงในลักษณะนี้จึงยังไม่เป็นผลการตรวจสอบที่แม่นยำเท่าใดนัก

และ จากที่ได้เคยกล่าวมาแล้ว ในขั้นตอนการทำ greedy string tiling ซึ่งเป็นการใช้ nested loop ซ้อนกัน 3 ชั้น ซึ่ง จะส่งผลอย่างมากกับเวลาในการ ใช้ตรวจสอบ โดยเฉพาะ ยิ่งใช้กับ การ ตรวจสอบข้อมูลที่มีความยาวของ source code มาก ดังนั้น การใช้ JPlag กับการตรวจสอบ source code ที่มีความยาว ค่อนข้างมาก จึงเป็นการสร้างข้อจำกัดในด้านเวลา ของ JPlag มากยิ่งขึ้นไปด้วย

ซึ่งจากตรงนี้ เราก็สามารถ นำเอา ระบบการตรวจสอบแบบ JPlag มาช่วยพัฒนา และปรับปรุง ผลการตรวจสอบ ของระบบที่ มีจุดด้อย ต่อการ เปลี่ยนแปลง ชนิดของ ตัวแปร หรือ การจัดวาง layout ของ source code ซึ่งเป็นจุดแข็งของ JPlag เพื่อให้ ได้ผลการตรวจสอบที่แม่นยำ

3.2.2.1 การใช้งานร่วมกันระหว่าง Jplag และ PDetect

Jplag มีความสามารถในการตรวจจับข้อมูลที่มีการปรับแต่ง Keyword แต่ขาดความแม่นยำกับข้อมูลที่สลับที่ ซึ่ง Pdetect มีความสามารถในการตรวจจับข้อมูลที่มีการสลับที่ได้ แต่ขาดความแม่นยำกับข้อมูลที่มีการปรับแต่ง Keyword ทำให้ทั้งสองวิธีสามารถลบจุดอ่อนซึ่งกันและกันได้ ข้อมูลซึ่งผ่านการกรองของทั้งสองวิธีจึงควรจะมีความแม่นยำที่มากขึ้น
การพัฒนา ตัวโปรแกรมจากทฤษฎีเดิม

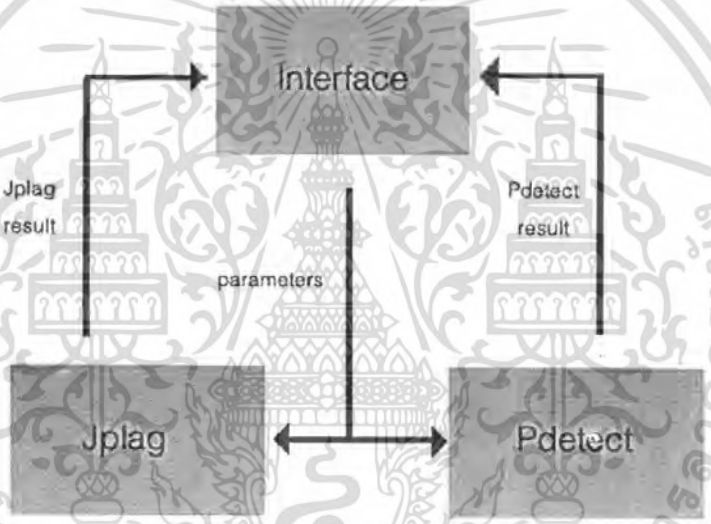
3.2.2.2 การพัฒนาเพิ่มเติม

จากเดิม ตัว JPlag ใช้ วิธีการตรวจสอบ โดยวิธีการหา longest common subsequence ซึ่งใช้ nested loop 3 ชั้น ส่งผลให้ big Oh เป็น n^3 และ มีจุดอ่อนอย่างยิ่งกับการ ถูกแทรก ด้วย redundancy หรือ การเปลี่ยนแปลง sequence ของโปรแกรม เพียงน้อยนิด อีกทั้งยังเป็นวิธีการที่สิ้นเปลืองเวลาเป็นอย่างมาก ซึ่งจากตรงนี้ จึงได้พัฒนา เปลี่ยน algorithm ที่ใช้ในการตรวจสอบ ใหม่ เป็นการ mapping token แทน ซึ่งให้ผลใกล้เคียงกับการทำ greedy string tilling โดยจะใช้ nested loop เพียง 2 ชั้นส่วนช่วยเช็คกรณีเกิด sub string ของ token จะสามารถ ผ่านไปยังต่อแปร ลำดับต่อไปของ loop ขั้นบนสุดได้ โดยไม่ต้องเสียเวลา ลงไป ใน loop อีก ซึ่งทำให้ big Oh ลดลงเหลือ n^2แต่อย่างไรก็ตาม แม้จะทำให้โปรแกรม เร็วขึ้น แต่ก็ ส่งผลต่อการตรวจสอบของโปรแกรมคือ จะทำให้ ค่าที่ได้จาก การตรวจสอบผ่านโปรแกรมนั้นมีค่าสูงขึ้นกว่า algorithm เดิม ในทุกกรณีแต่แนวโน้ม ของผลการตรวจสอบ จะยังคงแนวโน้มเดิม

# บทที่ 4
# การออกแบบและพัฒนา

## 4.1 โครงสร้างโปรแกรม

ออกแบบให้อินเตอร์เฟสเป็นส่วนเรียกโปรแกรมเอกเทศที่อยู่ภายนอกให้ทำงาน และรับเอาค่าที่ได้จากโปรแกรมเอกเทศทั้งสองนั้นมาแสดงผล

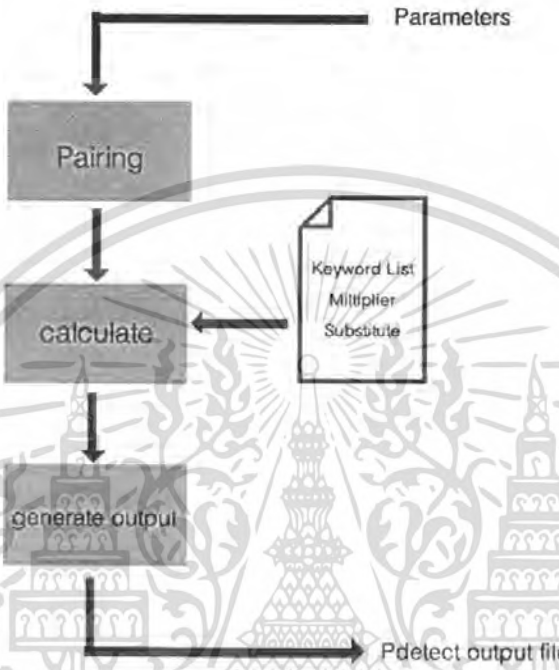



**ภาพที่ 4.1 : โครงสร้างของโปรแกรมโดยรวม**

### 4.1.1 ส่วนติดต่อกับผู้ใช้ (Interface)

- รับโฟลเดอร์ที่กำหนดจากผู้ใช้ และส่งให้กับโปรแกรมที่เหลือเป็นพารามิเตอร์
- ควบคุมด้วยเมาส์
- รับค่าที่ได้จากทั้งสองโปรแกรมมาแสดงผลให้ผู้ใช้ทราบ

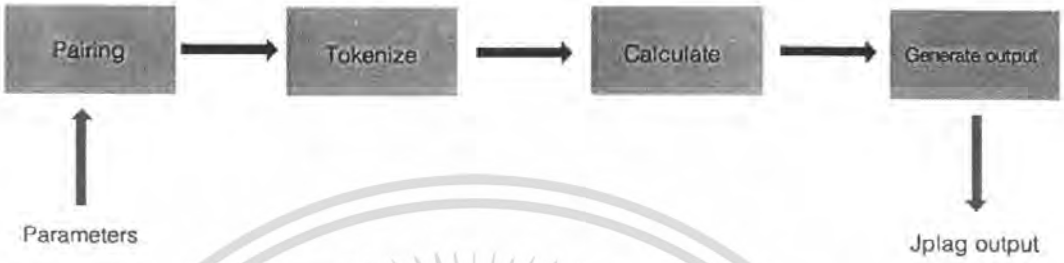
### 4.1.2 ส่วนโปรแกรม Pdetect




**ภาพที่ 4.2 : โครงสร้างของ PDetect**

- รับพารามิเตอร์เป็นชื่อที่เก็บไฟล์ที่ต้องการตรวจสอบจากส่วนติดต่อผู้ใช้
- ส่วนแพริ่ง (Pairing) จับคู่ไฟล์ทุกไฟล์และส่งให้ส่วนคำนวณ
- ส่วนคำนวณ ทำหน้าที่คำนวณตามหลักทฤษฎีของ Pdetect โดยใช้ไฟล์ภายนอกคือคีย์เวิร์ด ตัวคูณ และตัวแทนค่า มาใช้ร่วมด้วย
- ส่วนสร้างผล ทำหน้าที่สร้างไฟล์ที่บรรจุผลที่ได้ เพื่อให้ส่วนติดต่อผู้ใช้นำไปแสดงผล

### 4.1.3 ส่วนโปรแกรม Jplag




**ภาพที่ 4.3 : โครงสร้างของ Jplag**

- ส่วนแพริ่ง (Pairing) จับคู่ไฟล์ทุกไฟล์และส่งให้ส่วน โทเคไนซ์ (Tokenize)
- ส่วนโทเคไนซ์ ทำหน้าที่เปลี่ยนไฟล์เป็นโทเคน (Token) เพื่อใช้ในส่วนคำนวณ
- ส่วนคำนวณ ทำการหาค่าความเหมือนโดยใช่ทฤษฎีของ Jplag และส่งให้ส่วนสร้างผล
- ส่วนสร้างผล ทำหน้าที่สร้างไฟล์ที่บรรจุผลที่ได้ เพื่อให้ส่วนติดต่อผู้ใช้นำไปแสดงผล

# บทที่ 5

# การทดลอง

## 5.1 วิธีการทดลอง

การทดลองเพื่อใช้ในการตรวจสอบและช่วยในการสรุปผลของตัวโปรแกรม มีดังนี้

1. การตรวจสอบ โดยใช้ โปรแกรม 5 ตัวอย่าง
2. การตรวจสอบ โดยใช้ โปรแกรม 10 ตัวอย่าง
3. การตรวจสอบ โดยใช้ โปรแกรม 50 ตัวอย่าง
4. การตรวจสอบ โดยละเอียด ด้วยตามนุษย์ 5 ตัวอย่าง
6. การตรวจสอบ โดยละเอียด ด้วยตามนุษย์ 10 ตัวอย่าง

## 5.2 ผลการทดลองที่ได้

จากการทดลอง และตรวจสอบ โดยละเอียด โดยใช้ตัวอย่าง 10 ตัวอย่าง
ซึ่งค่าที่ได้ออกมา จะมี n x (n-1) ค่า หรือ 10 x 9 = 90 ค่า
ทำให้สามารถตีความหมายของค่าตัวเลขออกมาเป็น 2 กลุ่มได้ดังนี้

- กลุ่มที่ให้ผลเป็นลบ
ค่าที่ได้ ต่ำกว่า 70% มีโอกาสต่ำมากที่จะเป็นโปรแกรมที่จะลอกกันมา

- กลุ่มที่ให้ผลเป็นบวก
ค่าที่ได้ มากกว่า 70% สามารถตั้งข้อสงสัยได้ว่า มีโอกาสที่จะเป็นโปรแกรมที่ลอกกันมา
และ สามารถ แบ่งผลการตรวจสอบ ออกมาได้ เป็น 4 กลุ่ม คือ

ตารางที่ 5.1 แสดงผลการทดลองสิบตัวอย่าง

**5.2.1 กลุ่มที่ 1 ทั้ง PDetect และ JPlag ให้ผล เป็น บวก ทั้งคู่**

[code no 1] =pair with= [code no 3] = Value : 84.2105

[ code no 1 ] =pair with= [ code no 3 ] value = 82

[code no 3] =pair with= [code no 1] = Value : 84.2105

[ code no 3 ] =pair with= [ code no 1 ] value = 92

[code no 3] =pair with= [code no 4] = Value : 100

[ code no 3 ] =pair with= [ code no 4 ] value = 72

[code no 3] =pair with= [code no 5] = Value : 73.6842

[ code no 3 ] =pair with= [ code no 5 ] value = 72

[code no 3] =pair with= [code no 6] = Value : 100

[ code no 3 ] =pair with= [ code no 6 ] value = 72

[code no 3] =pair with= [code no 7] = Value : 100

[ code no 3 ] =pair with= [ code no 7 ] value = 72

[code no 3] =pair with= [code no 8] = Value : 100

[ code no 3 ] =pair with= [ code no 8 ] value = 72

[code no 3] =pair with= [code no 9] = Value : 78.9474

[ code no 3 ] =pair with= [ code no 9 ] value = 72

[code no 4] =pair with= [code no 6] = Value : 100

[ code no 4 ] =pair with= [ code no 6 ] value = 100

[code no 4] =pair with= [code no 7] = Value : 96.9697

[ code no 4 ] =pair with= [ code no 7 ] value = 100

[code no 4] =pair with= [code no 8] = Value : 100

[ code no 4 ] =pair with= [ code no 8 ] value = 100

[code no 5] =pair with= [code no 1] = Value : 100

[ code no 5 ] =pair with= [ code no 1 ] value = 85

[code no 5] =pair with= [code no 3] = Value : 100

[ code no 5 ] =pair with= [ code no 3 ] value = 77

[code no 5] =pair with= [code no 4] = Value : 100

[ code no 5 ] =pair with= [ code no 4 ] value = 100

[code no 5] =pair with= [code no 6] = Value : 100

[ code no 5 ] =pair with= [ code no 6 ] value = 100

[code no 5] =pair with= [code no 7] = Value : 100

[ code no 5 ] =pair with= [ code no 7 ] value = 100

[code no 5] =pair with= [code no 8] = Value : 100

[ code no 5 ] =pair with= [ code no 8 ] value = 100

[code no 5] =pair with= [code no 9] = Value : 100

[ code no 5 ] =pair with= [ code no 9 ] value = 94

[code no 5] =pair with= [code no10] = Value : 100

[ code no 5 ] =pair with= [ code no10 ] value = 88

[code no 6] =pair with= [code no 4] = Value : 100

[ code no 6 ] =pair with= [ code no 4 ] value = 100

[code no 6] =pair with= [code no 7] = Value : 96.9697

[ code no 6 ] =pair with= [ code no 7 ] value = 100

[code no 6] =pair with= [code no 8] = Value : 100

[ code no 6 ] =pair with= [ code no 8 ] value = 100

[code no 7] =pair with= [code no 4] = Value : 100

[ code no 7 ] =pair with= [ code no 4 ] value = 100

[code no 7] =pair with= [code no 6] = Value : 100

[ code no 7 ] =pair with= [ code no 6 ] value = 100

[code no 7] =pair with= [code no 8] = Value : 100

[ code no 7 ] =pair with= [ code no 8 ] value = 100

[code no 8] =pair with= [code no 4] = Value : 100

[ code no 8 ] =pair with= [ code no 4 ] value = 100

[code no 8] =pair with= [code no 6] = Value : 100

[ code no 8 ] =pair with= [ code no 6 ] value = 100

[code no 8] =pair with= [code no 7] = Value : 96.9697

[ code no 8 ] =pair with= [ code no 7 ] value = 100

[code no 9] =pair with= [code no 1] = Value : 100

[ code no 9 ] =pair with= [ code no 1 ] value = 92

[code no 9] =pair with= [code no 3] = Value : 100

[ code no 9 ] =pair with= [ code no 3 ] value = 78

[code no 9] =pair with= [code no 4] = Value : 100

[ code no 9 ] =pair with= [ code no 4 ] value = 100

[code no 9] =pair with= [code no 5] = Value : 93.3333

[ code no 9 ] =pair with= [ code no 5 ] value = 100

[code no 9] =pair with= [code no 6] = Value : 100

[ code no 9 ] =pair with= [ code no 6 ] value = 100

[code no 9] =pair with= [code no 7] = Value : 100

[ code no 9 ] =pair with= [ code no 7 ] value = 100

[code no 9] =pair with= [code no 8] = Value : 100

[ code no 9 ] =pair with= [ code no 8 ] value = 100

[code no 9] =pair with= [code no10] = Value : 93.3333

[ code no 9 ] =pair with= [ code no10 ] value = 89

[code no10] =pair with= [code no 3] = Value : 77.2727

[ code no10 ] =pair with= [ code no 3 ] value = 70

[code no10] =pair with= [code no 4] = Value : 95.4545

[ code no10 ] =pair with= [ code no 4 ] value = 90

[code no10] =pair with= [code no 6] = Value : 95.4545

[ code no10 ] =pair with= [ code no 6 ] value = 90

[code no10] =pair with= [code no 7] = Value : 95.4545

[ code no10 ] =pair with= [ code no 7 ] value = 90

[code no10] =pair with= [code no 8] = Value : 95.4545

[ code no10 ] =pair with= [ code no 8 ] value = 90

**5.2.2 กลุ่มที่ 2 Pdetect ให้ผลเป็น บวก แต่ JPlag ให้ผลเป็น ลบ**

[code no 1] =pair with= [code no 2] = Value : 84.2105

[ code no 1 ] =pair with= [ code no 2 ] value = 56

[code no 1] =pair with= [code no 4] = Value : 84.2105

[ code no 1 ] =pair with= [ code no 4 ] value = 69

[code no 1] =pair with= [code no 5] = Value : 73.6842

[ code no 1 ] =pair with= [ code no 5 ] value = 69

[code no 1] =pair with= [code no 6] = Value : 84.2105

[ code no 1 ] =pair with= [ code no 6 ] value = 69

[code no 1] =pair with= [code no 7] = Value : 84.2105

[ code no 1 ] =pair with= [ code no 7 ] value = 69

[code no 1] =pair with= [code no 8] = Value : 84.2105

[ code no 1 ] =pair with= [ code no 8 ] value = 69

[code no 1] =pair with= [code no 9] = Value : 78.9474

[ code no 1 ] =pair with= [ code no 9 ] value = 69

[code no 1] =pair with= [code no10] = Value : 78.9474

[ code no 1 ] =pair with= [ code no10 ] value = 65

[code no 3] =pair with= [code no 2] = Value : 89.4737

[ code no 3 ] =pair with= [ code no 2 ] value = 60

[code no 3] =pair with= [code no10] = Value : 89.4737

[ code no 3 ] =pair with= [ code no10 ] value = 52

[code no 4] =pair with= [code no 2] = Value : 81.8182

[ code no 4 ] =pair with= [ code no 2 ] value = 39

[code no 5] =pair with= [code no 2] = Value : 100

[ code no 5 ] =pair with= [ code no 2 ] value = 54

[code no 6] =pair with= [code no 2] = Value : 81.8182

[ code no 6 ] =pair with= [ code no 2 ] value = 39

[code no 7] =pair with= [code no 2] = Value : 84.375

[ code no 7 ] =pair with= [ code no 2 ] value = 45

[code no 8] =pair with= [code no 2] = Value : 81.8182

[ code no 8 ] =pair with= [ code no 2 ] value = 39

[code no 9] =pair with= [code no 2] = Value : 93.3333

[ code no 9 ] =pair with= [ code no 2 ] value = 57

[code no10] =pair with= [code no 2] = Value : 95.4545

[ code no10 ] =pair with= [ code no 2 ] value = 30

### 5.2.3 กลุ่มที่ JPlag ให้ผล เป็นบวก แต่ PDetect ให้ผล เป็น ลบ

[ code no 4 ] =pair with= [ code no 5 ] value = 100

[code no 4] =pair with= [code no 5] = Value : less than 50%

[ code no 4 ] =pair with= [ code no 9 ] value = 80

[code no 4] =pair with= [code no 9] = Value : less than 50%

[ code no 4 ] =pair with= [ code no10 ] value = 84

[code no 4] =pair with= [code no10] = Value : less than 50%

[ code no 6 ] =pair with= [ code no 5 ] value = 100

[code no 6] =pair with= [code no 5] = Value : less than 50%

[ code no 6 ] =pair with= [ code no 9 ] value = 80

[code no 6] =pair with= [code no 5] = Value : less than 50%

[ code no 6 ] =pair with= [ code no10 ] value = 84

[code no 6] =pair with= [code no10] = Value : 63.6364

[ code no 7 ] =pair with= [ code no 5 ] value = 100

[code no 7] =pair with= [code no 5] = Value : less than 50%

[ code no 7 ] =pair with= [ code no 9 ] value = 81

[code no 7] =pair with= [code no 9] = Value : less than 50%

[ code no 7 ] =pair with= [ code no10 ] value = 81

[code no 7] =pair with= [code no10] = Value : 65.625

[ code no 8 ] =pair with= [ code no 5 ] value = 100

[code no 8] =pair with= [code no 5] = Value : less than 50%

[ code no 8 ] =pair with= [ code no 9 ] value = 80

[code no 8] =pair with= [code no 9] = Value : less than 50%

[ code no 8 ] =pair with= [ code no10 ] value = 84

[code no 8] =pair with= [code no10] = Value : 63.6364

[ code no10 ] =pair with= [ code no 1 ] value = 80

[code no10] =pair with= [code no 1] = Value : 68.1818

[ code no10 ] =pair with= [ code no 5 ] value = 90

[code no10] =pair with= [code no 5] = Value : 63.6364

[ code no10 ] =pair with= [ code no 9 ] value = 80

[code no10] =pair with= [code no 9] = Value : 63.6364

**5.2.4 กลุ่มที่ ทั้ง PDetect และ JPlag ให้ผล เป็น ลบ**

[ code no 2 ] =pair with= [ code no 1 ] value = 53

[code no 2] =pair with= [code no 1] = Value : less than 50

[ code no 2 ] =pair with= [ code no 3 ] value = 65

[code no 2] =pair with= [code no 3] = Value : less than 50

[ code no 2 ] =pair with= [ code no 4 ] value = 56

[code no 2] =pair with= [code no 4] = Value : less than 50

[ code no 2 ] =pair with= [ code no 5 ] value = 56

[code no 2] =pair with= [code no 5] = Value : less than 50

[ code no 2 ] =pair with= [ code no 6 ] value = 56

[code no 2] =pair with= [code no 6] = Value : 67.5

[ code no 2 ] =pair with= [ code no 7 ] value = 56

[code no 2] =pair with= [code no 7] = Value : 67.5

[ code no 2 ] =pair with= [ code no 8 ] value = 56

[code no 2] =pair with= [code no 8] = Value : 67.5

[ code no 2 ] =pair with= [ code no 9 ] value = 56

[code no 2] =pair with= [code no 9] = Value : less than 50

[ code no 2 ] =pair with= [ code no10 ] value = 18

[code no 2] =pair with= [code no10] = Value : 52.5

[ code no 3 ] =pair with= [ code no 2 ] value = 60

[code no 3] =pair with= [code no 2] = Value : less than 50

[ code no 3 ] =pair with= [ code no10 ] value = 52

[code no 3] =pair with= [code no10] = Value : less than 50

[ code no 4 ] =pair with= [ code no 1 ] value = 68

[code no 4] =pair with= [code no 1] = Value : less than 50

[ code no 4 ] =pair with= [ code no 3 ] value = 68

[code no 4] =pair with= [code no 3] = Value : 57.5758

[ code no 6 ] =pair with= [ code no 1 ] value = 68

[code no 6] =pair with= [code no 1] = Value : less than 50

[ code no 6 ] =pair with= [ code no 3 ] value = 68

[code no 6] =pair with= [code no 3] = Value : 57.5758

[ code no 7 ] =pair with= [ code no 1 ] value = 67

[code no 7] =pair with= [code no 1] = Value : 50

[ code no 7 ] =pair with= [ code no 3 ] value = 67

[code no 7] =pair with= [code no 3] = Value : 59.375

[ code no 8 ] =pair with= [ code no 1 ] value = 68

[code no 8] =pair with= [code no 1] = Value : less than 50

[ code no 8 ] =pair with= [ code no 3 ] value = 68

[code no 8] =pair with= [code no 3] = Value : 57.5758

### 5.3. วิเคราะห์ผลการทดลอง

จากการทดลองข้างต้น ได้ใช้ คนเข้ามาตรวจ ตัวโปรแกรมซ้ำอีกครั้งหนึ่งและเทียบเคียงผลที่ได้จากการตรวจด้วยคนกับผลจากโปรแกรม พบว่า

- โปรแกรม หมายเลข 1 กับ 3 มีความคล้ายกัน ด้าน structure และ จำนวน keyword อยู่ในระดับหนึ่ง
แต่ ลักษณะ การทำงาน แตกต่างกันออกไป
- โปรแกรมหมายเลข 2 ทั้งตัว structure และ keyword รวมถึงลักษณะ การใช้งาน รูปแบบการเขียน
แตกต่างจาก โปรแกรม หมายเลขอื่นๆ
- โปรแกรม หมายเลข 4, 6, 7 และ 8 เป็นกลุ่มที่ลอก กันมา กลุ่มที่ 1 และ โปรแกรม หมายเลข 5 กับ 9 เป็นกลุ่มที่ลอกกันมา กลุ่มที่ 2 โดยความคล้ายกันของ structure และจำนวน keyword ทำให้สามารถ สรุปความหมายของค่าตัวเลขที่ได้ ในแต่ละกลุ่มดังนี้

1. กลุ่มที่ให้ผล เป็นบวก ทั้งคู่ ตัวโปรแกรม มีความคล้ายกัน ทั้งด้าน structure และ flow control และ มีจำนวน Keyword ใกล้เคียงกัน

2. กลุ่มที่ PDetect ให้ผลเป็นบวก แต่ JPlag ให้ผลเป็นลบ ตัวโปรแกรม มีจำนวน Keyword ใกล้เคียงกัน แต่ลักษณะ ของ structure และ flow control แตกต่างกัน สรุป ได้ว่า มีโอกาสน้อยมากที่ จะเป็นโปรแกรมที่ลอกกันมา

3. กลุ่มที่ JPlag ให้ผลเป็นบวก แต่ PDetect ให้ผลเป็นลบ ตัวโปรแกรม มี Structure คล้ายกัน แต่ keyword ต่างกัน เป็นไปได้ว่า อาจจะใช้แนวคิดใกล้เคียงกัน แต่การออกแบบโปรแกรม แตกต่างกัน จึงสรุปว่า มีโอกาส น้อยมากที่จะเป็นโปรแกรมที่ลอกกันมา

4. กลุ่มที่ทั้ง JPlag และ PDetect ให้ผล เป็นลบ ตัวโปรแกรม มีความแตกต่างกันทั้ง Structure และ keyword จึงสรุปว่า ไม่น่าจะเป็น code ที่ลอกกันมา

## 5.4. สรุปผลการทดลอง

จากการทดลอง ทั้งการตรวจสอบด้วยตัวโปรแกรม และ การตรวจสอบด้วยคน ทำให้ เมื่อเทียบเคียงค่าที่ได้จาก ตัวโปรแกรม และ ผลการตรวจด้วยตามนุษย์แล้ว ทำให้สามารถ สรุป ผลการตรวจสอบโดย แบ่งค่าของตัวเลขออกมาเป็นช่วงต่างๆ ได้ดังนี้

1. การเทียบ จาก code a เทียบกับ code b และ code b เทียบกับ code a
ทั้ง PDetect และ JPlag ให้ค่าออกมา มากกว่า 92% ในการเทียบทั้ง 2 แบบ สรุปได้ว่า
code a และ b เป็น code ที่ลอกกันมา

2. การเทียบ จาก code a เทียบกับ code b หรือ code b เทียบกับ code a
ทั้ง PDetect และ JPlag ให้ค่าออกมา มากกว่า 92% ในการเทียบแบบใดครั้งหนึ่ง สรุปได้ว่า
code a หรือ b น่าสงสัยว่า เป็น code ที่ลอกกันมา หรือ อาจจะมี structure ใกล้เคียงกันเท่านั้น

3. การเทียบ จาก code a เทียบกับ code b และ code b เทียบกับ code a
ทั้ง PDetect และ JPlag ให้ผลออกมาว่า มีโอกาส ปานกลาง และ/หรือ เล็กน้อย ในการเทียบ ทั้ง 2 แบบ
สรุปว่า อาจจะมีบางเพียงบางส่วนของ code a กับ b ที่คล้ายกัน

4. การเทียบ จาก code a เทียบกับ code b และ code b เทียบกับ code a
ทั้ง PDetect และ Jplag ให้ผลออกมา เป็นลบ ในการ เปรียบเทียบ ครั้งใดครั้งหนึ่ง หรือ มากกว่า
สรุปว่า ตัวโปรแกรม อาจจะมีบางส่วนคล้ายกันเท่านั้น ไม่น่าจะเป็น code ที่ลอกกันมา

**กรณี การตรวจสอบผิดพลาดของตัวโปรแกรม**

จากการทดลองพบว่า ความผิดพลาดในการตรวจสอบนั้น เกิดขึ้นในลักษณะ ที่ พบว่า คู่โปรแกรม มีโอกาสเป็นไปได้มาก ที่จะลอก กันมา แต่ คนเขียน ไม่ได้ลอกกันมา โดยมีสาเหตุ คือ

- ผู้เขียน โปรแกรม ได้ใช้แนวคิด และวิธีการออกแบบโปรแกรม มาคล้ายกัน หรือ อาจจะได้ รับคำแนะนำ มาแบบเดียวกัน จึงทำให้ ส่วนของโปรแกรม ออกมาเหมือนกัน มาก
- ตัวโปรแกรม ที่มี structure คล้ายกัน จำนวน keyword ใกล้เคียงกัน เขียนโดยคนเดียวกัน แต่ทำงานคนละอย่าง กัน สามารถให้ผลออกมา ใกล้เคียงกันได้ ดังนั้น จึงไม่ควรใช้ ตรวจสอบโปรแกรม ที่ ใช้งานต่างรูปแบบกัน

**ตารางที่ 5.2 ตารางแสดงความถูกต้องของ การตรวจสอบของโปรแกรม**

| | True Positive | True Negative | False Positive | False Negative |
|---|---|---|---|---|
| การทดลอง 5 ตัวอย่าง | 0 | 5 | 0 | 0 |
| การทดลอง 10 ตัวอย่าง | 6 | 4 | 0 | 0 |
| การทดลอง 36 ตัวอย่าง | 12 | 23 | 1 | 0 |
| การทดลอง 50 ตัวอย่าง | 19 | 28 | 3 | 0 |
| รวม % | | | 4.65% | 0% |

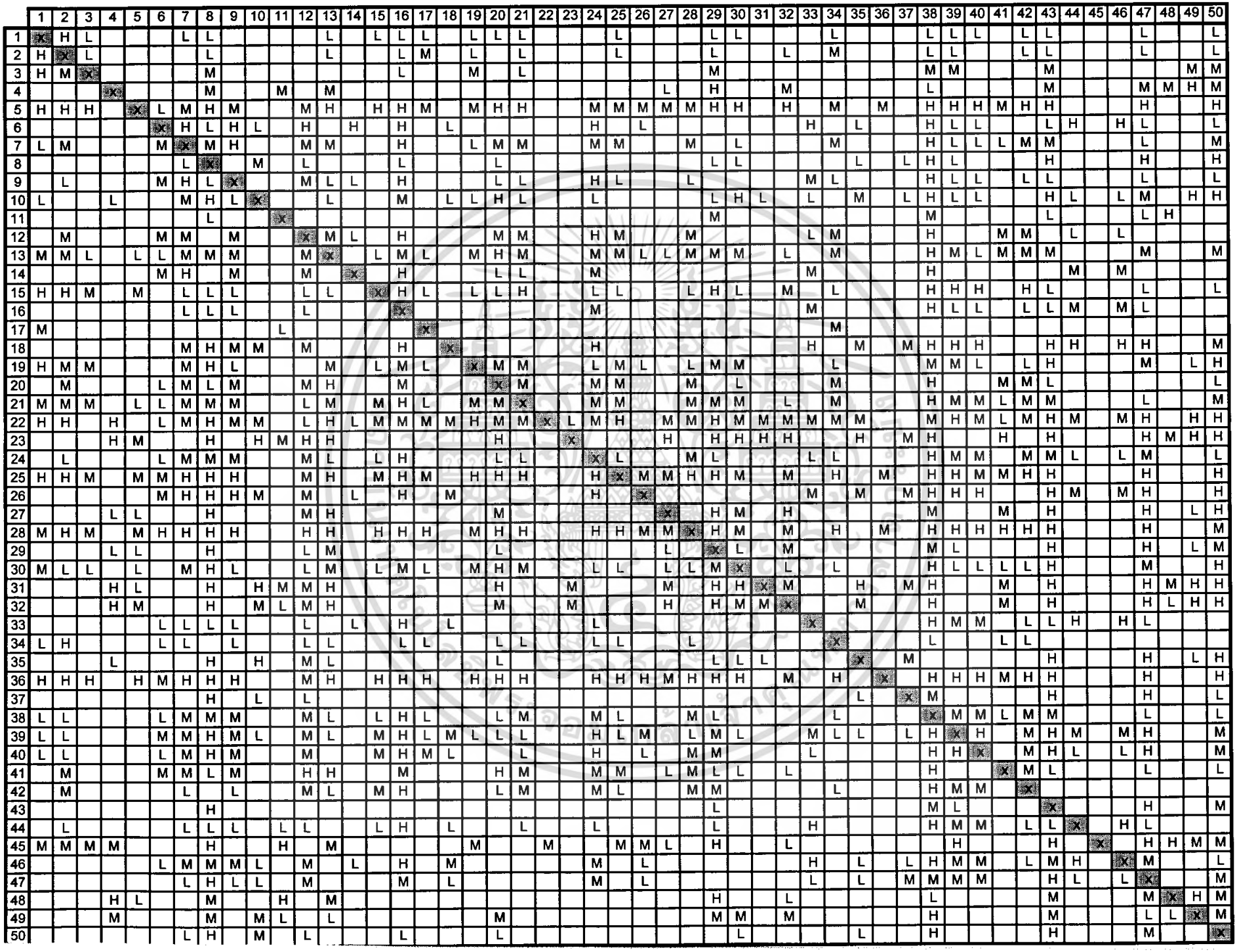
โดยค่าในตาราง มีค่าเป็นจำนวนตัวอย่าง และแต่ละช่องมีความหมายดังนี้

True Positive คือ การตรวจสอบแล้ว พบว่า เป็นโปรแกรมที่ลอกกันมาจริง

True Negative คือการตรวจสอบแล้ว พบว่า เป็นโปรแกรมที่ไม่ได้ลอกกันมาจริง

False Positive คือ ตัวโปรแกรม แจ้งว่า เป็นโปรแกรมที่ลอกกันมา แต่ อันที่จริงแล้ว ไม่ได้ลอกกันมา

False Negative คือ ตัวโปรแกรม แจ้งว่าไม่ได้เป็นโปรแกรมที่ลอกกันมา แต่ อันที่จริงแล้ว ลอกกันมา



# บรรณานุกรม


นิรุธ อำนวยศิลป์, 2005.**Visual C++ and MFC Programming.** ดวงกมลสมัย.

ประยงค์ อู่ประสิทธิ์วงศ์, 2006. **หลักการเขียนโปรแกรมและการแก้ปัญหาด้วยภาษา C++.** ดวงกมลสมัย.

สุรสิทธิ์ คิวประสพศักดิ์ และนันทนี แขวงโสภา, 2003. **อินไซท์ Visual Basic .NET ฉบับสมบูรณ์.** โปรวิชั่น.

CPlusPlus.com, 2000 – 2008. **String – C++ Reference.** [Online] Avaliable : http://www.cplusplus.com/reference/string/string/

L. Moussiades and A. Vakali, 2005. **PDetect: A Clustering Approach for Detecting Plagiarism in Source Code Datasets.** Oxford University Press.

Prechelt, L., Malpohl, G. and Philippsen M. , 2002. **Finding plagiarisms among a set of programs with Jplag.** J. Univ. Comput. Sci.

# ภาคผนวก ก. ซอร์สโค้ดและอินเตอร์เฟส

การพัฒนาโปรแกรมนี้เริ่มด้วยการนำอัลกอริธึมของ Pdetect และ Jplag นำมาสร้างเป็นโปรแกรมใช้งานจริง โดยมีซอร์สโค้ดดังนี้

**ซอร์สโค้ดของ PDetect**

ฟังก์ชั่น Cleanstr

```
int cleanstr(char* path, char* pathout)// CLEAN STRING
        {
        char * buffer;
        int _CountBuffer;A
        _CountBuffer = 0 ;
        ifstream inputStream;
        ofstream outputStream;

        inputStream.open(path,ios::binary);

        if(inputStream.fail())
        {
                cerr << "*** ERROR: Cannot open" << path
                        << " for input." << endl;
                return EXIT_FAILURE ;
        }

        inputStream.seekg(0,ios::end);
        _CountBuffer = inputStream.tellg();
        inputStream.seekg(0,ios.beg);
```

```
buffer = new char [_CountBuffer];


inputStream.read (buffer,_CountBuffer);
inputStream.close();


/*************** Start Control Buffer ********************/
vector<char> BufferBeforeOut;
bool _IsStop;
int indexCopy;
int indexBuffer;
_IsStop = true;
indexBuffer = 0;
indexCopy = 0;

while(indexCopy < _CountBuffer) //if index less than last char do while loop
{
      if((buffer[indexCopy] == '/') && (buffer[indexCopy + 1] == '/'))
      {
            while (_IsStop)
            {
                  indexCopy = indexCopy + 1;
                  if(buffer[indexCopy] == '\n')
                  {
                        _IsStop = false;
                        indexCopy = indexCopy + 1;
                  }
            }
            _IsStop = true;
      }
```

```
        else if ((buffer[indexCopy] == '/') && (buffer[indexCopy + 1] == '*'))
        {
                while (_IsStop)
                {
                        indexCopy = indexCopy + 1;
                        if((buffer[indexCopy] == '*') && (buffer[indexCopy + 1] == '/'))
                        {
                                _IsStop = false;
                                indexCopy = indexCopy + 2;
                        }
                }
                _IsStop = true;
        }
        else
        {
                BufferBeforeOut.push_back(buffer[indexCopy]);
                indexCopy = indexCopy + 1;
                indexBuffer = indexBuffer + 1;
        }
}

/************* Start Write To File ****************/
char *buf;
buf = new char [BufferBeforeOut.size()];
for(int index = 0; index < (BufferBeforeOut.size());index++)
{
        buf[index] = BufferBeforeOut[index];
}
```

```
        outputStream.open(pathout,ios::binary);
        outputStream.write(buf,BufferBeforeOut.size());
        outputStream.close();

        return 0;
}
```

ฟังก์ชั่นนี้ใช้ลบคอมเม้นท์ออกจากตัวโค้ด โดยจะทำการอ่านทีละตัวลงไปในบัฟเฟอร์ และตรวจหาสัญลักษณ์ของการคอมเม้นท์ทั้งแบบบรรทัดเดียวและหลายบรรทัดไปด้วย โดยเมื่อพบแล้ว ตัวแปลแฟลก (Flag) จะถูกยกขึ้น ทำให้ไม่มีการอ่านส่วนคอมเม้นท์จากส่วนอ่านลงไปในบัฟเฟอร์ และเมื่อเสร็จแล้วก็จะทำการเขียนทุกอย่างในบัฟเฟอร์ลงบนไฟล์ชั่วคราวที่จะใช้ในการประมวลผลในโปรแกรมส่วนต่อไป

ฟังก์ชั่น Searchstr

```
        int searchstr(string str, string code){ // search string from listfile
                size_t foundpos = 0;
                int counter = 0;
                while (foundpos != string::npos) {
                counter++;
                foundpos = code.find(str,foundpos+1);
                }
                return counter-1;
        }
```

ใช้หา String ที่กำหนดจากชุดตัวแปรที่บรรจุโค้ดรวมเอาไว้โดยจะระบุตำแหน่งของอักษรแรกที่หาพบ

Struct Table

```
struct tables{
        string name;
        int count;
};
```

ใช้เป็นตัวแปรของอาร์เรย์สำหรับบรรจุค่าคีย์เวิร์ดและจำนวนครั้งที่ตรวจพบคีย์เวิร์ดในโปรแกรม

ฟังก์ชั่น PDetect

```
double pdetect(char* file1_path, char* file2_path)
{
        string line;
        string allcode1;
        string allcode2;
        ifstream infile1;
        ifstream infile2;
        ifstream mulfile;
        ifstream subsfile;
        ifstream listfile1;
        ifstream listfile2;
        string searchnow;
        tables tab1[50];
        tables tab2[50];
        int arraycount1 = 0;
        int arraycount2 = 0;
        int seecount;

        cleanstr(file1_path, "file1_c.txt");
        cleanstr(file2_path, "file2_c.txt");
```

```
        infile1.open("file1_c.txt");          // append file into string
        while (! infile1.eof()){
                getline(infile1,line);
                line.append("??");
                allcode1.append(line);
        }
        infile1.close();

        infile2.open("file2_c.txt");
        while (! infile2.eof()){
                getline(infile2,line);
                line.append("??");
                allcode2.append(line);
        }
        infile2.close();


        listfile1.open("list.txt"); // BUILD ARRAY FILE1
        while (! listfile1.eof()){
                getline(listfile1,searchnow);
        seecount = searchstr(searchnow,allcode1);
        tab1[arraycount1].name = searchnow;
        tab1[arraycount1].count = seecount;
        seecount = 0;
        arraycount1++;
        }
listfile1.close();
```

```
        listfile2.open("list.txt"); // BUILD ARRAY FILE2
        while (! listfile2.eof()){
                getline(listfile2,searchnow);
        seecount = searchstr(searchnow,allcode2);
        tab2[arraycount2].name = searchnow;
        tab2[arraycount2].count = seecount;
        seecount = 0;
        arraycount2++;
        }
listfile2.close();

mulfile.open("multiplier.txt"); // MULTIPLY
        while (! mulfile.eof()){
                string multxt;
                getline(mulfile, multxt);
int mulnum = *(multxt.end()-1)-48;
multxt.erase(multxt.end()-2, multxt.end());

        for (int i=0; i < arraycount1; i++){
                if (tab1[i].name == multxt){
                        tab1[i].count = tab1[i].count * mulnum;}
        }
        for (int i=0; i < arraycount2; i++){
                if (tab2[i].name == multxt) {
                        tab2[i].count = tab2[i].count * mulnum;}
        }
}
mulfile.close();
```

```
subsfile.open("substitute.txt"); // SUBSTITUTE
        while (! subsfile.eof()){
                char subsname[15];
                string subschange;
                size_t foundblank;
                getline(subsfile, subschange);
                foundblank = subschange.find(" ");
                if (foundblank!=string::npos)
                subschange.copy(subsname, foundblank, 0);
                subsname[foundblank]='\0';
                subschange.erase(0, foundblank+1);

                for (int i=0; i < arraycount1; i++){
                        if (tab1[i].name == subsname) {
                                tab1[i].name = subschange;
                        }
        }
                for (int j=0; j < arraycount2; j++){
                        if (tab2[j].name == subsname) {
                                tab2[j].name = subschange;
                        }
        }

}
subsfile.close();

for (int i=0; i < arraycount1; i++) // BLANKOUT DUPLICATE TABLE1 ENTRY
{
        for (int j=0; j < arraycount1; j++)
```

```
        {
                if (i!=j){
                        if (tab1[i].name == tab1[j].name){
                                tab1[i].count = tab1[i].count + tab1[j].count;
                                tab1[j].count = 0;
                                tab1[j].name = "";
                        }
                }
        }
}

for (int i=0; i < arraycount2; i++) // BLANKOUT DUPLICATE TABLE2 ENTRY
{
        for (int j=0; j < arraycount2; j++)
        {
                if (i!=j){
                        if (tab2[i].name == tab2[j].name){
                                tab2[i].count = tab2[i].count + tab2[j].count;
                                tab2[j].count = 0;
                                tab2[j].name = "";
                        }
                }
        }
}
for (int i=0; i < arraycount1; i++) // DELETE BLANK ENTRY IN TABLE1
{
        if (tab1[i].name == "")
        {
                for (int j = i; j < arraycount1; j++){
```

```
                tab1[j].name = tab1[j+1].name;
                tab1[j].count = tab1[j+1].count;
                }
        arraycount1 = arraycount1 - 1;
        i = i-1;
        }
}
for (int i=0; i < arraycount2; i++) // DELETE BLANK ENTRY IN TABLE2
{
        if (tab2[i].name == "")
        {
                for (int j = i; j < arraycount2; j++){
                tab2[j].name = tab2[j+1].name;
                tab2[j].count = tab2[j+1].count;
                }
        arraycount2 = arraycount2 - 1;
        i = i-1;
        }
}

double base = 0; // CALCULATE PLAGIARISM PERCENTAGE

        for (int i=0; i < arraycount1; i++){
                base = base + tab1[i].count;
        }
double subs = 0;
i = 0;
int j = 0;
```

```
for (i=0; i < arraycount1; i++){
                for (j=0; j < arraycount2; j++){
                        if (tab1[i].name == tab2[j].name){
                                if (tab1[i].count < tab2[j].count){
                                        subs = subs + tab1[i].count;
                                }
                                else subs = subs + tab2[j].count;
                                break;
                        }
                }
        }
// OUTPUT

        return (subs/base) * 100;
}
```

เริ่มด้วยการอ่านไฟล์ที่กำหนดทั้งหมดไปใส่ในตัวแปรตัวแปรหนึ่ง ทั้งสองชุดที่จะเปรียบเทียบ ต่อมาทำการอ่านไฟล์ที่เก็บคีย์เวิร์ดที่ต้องการตรวจสอบนำมาใส่ไว้ในอาร์เรย์ของ Table เพื่อสร้างอาร์เรย์ที่จะใช้เก็บผลของจำนวนคีย์เวิร์ดที่ตรวจพบของทั้งสองไฟล์
และวนเรียกฟังก์ชั่น Searchstr ไปจนจบไฟล์เพื่อหาว่ามีคีย์เวิร์ดเกิดขึ้นกี่ครั้ง

หลังจากนั้นอ่านไฟล์ตัวคูณเพื่อนำไปคูณกับทุกตัวในอาร์เรย์ทั้งสองอันที่มีรายชื่ออยู่ในไฟล์ตัวคูณ โดยจะทำการคูณจำนวนครั้งที่พบที่เก็บในอาร์เรย์

ทำการอ่านไฟล์แทนค่าและนำค่าที่อ่านได้มาเปลี่ยนชื่อของตัวแปรทั้งหมดในอาร์เรย์อีกครั้ง เพื่อทำการลดรูปคีย์เวิร์ดและจัดกลุ่มคีย์เวิร์ดให้อยู่ในกลุ่มเดียวกัน ในตอนนี้อาร์เรย์ที่ใช้เก็บผลจะมีชื่อซ้ำกันอยู่ จึงต้องนำไปผ่านการวนที่จะทำการรวมชื่อทั้งหมดเข้าเป็นชื่อเดียวและนำจำนวนครั้งของทั้งหมดมารวมกัน ต่อไปก็จะทำการทำให้ช่องว่างที่เกิดขึ้นจากการรวมชื่อหายไป โดยดึงตัวที่อยู่ด้านบนลงมาปิดช่องว่างและลดตัวแปรที่บอกขนาดของอาร์เรย์ให้เหมาะสม

ในขั้นตอนสุดท้ายจะทำการคำนวณค่าความเหมือนกัน โดยขั้นแรก ตรวจว่าชื่อคีย์เวิร์ดตรงกันหรือไม่ และนำค่าครั้งที่พบของโปรแกรมที่สองมาเปรียบเทียบกับโปรแกรมแรก ทำการคำนวณผลต่างออกมา และนำจำนวนครั้งที่น้อยกว่าไปใส่ในตัวแปร Sub และนำค่าทั้งหมดของโปรแกรมแรกมาใส่ในตัวแปร base ทำการคำนวณหาเปอร์เซ็นต์ค่าความพลา-เกียไรซ์ (Plagiarize) ออกมาเป็นเปอร์เซ็นต์

ฟังก์ชั่น Main

```
int main(int argc, const char* argv[])
{
 const char* dir_path = argv[1];

WIN32_FIND_DATA f;
HANDLE h = FindFirstFile(dir_path, &f);
vector<string> filepath;

if(h != INVALID_HANDLE_VALUE){
        do{
                string filename = dir_path;
                filename.erase(filename.length()-1,1);
                filename.append(f.cFileName);
                if(filename.find(".cpp") != filename.npos){
                filepath.push_back(filename);
                }
        }
        while(FindNextFile(h, &f));
}
else{
fprintf(stderr, "Error opening directory\n");
}
```

```
// LOOP PAIRING


ofstream outfile;


outfile.open("output.txt");
for (int i= 0; i < filepath.size(); i++){
        for (int j = 0; j < filepath.size(); j++){
                if (filepath[i] != filepath[j]){


                        vector<char> v1(filepath[i].length() + 1);
                        strcpy(&v1[0], filepath[i].c_str());
                        char * charpath_a = &v1[0];

                        vector<char> v2(filepath[j].length() + 1);
                        strcpy(&v2[0], filepath[j].c_str());
                        char * charpath_b = &v2[0];
                        double results = pdetect(charpath_a, charpath_b);
                        if (results >= 50){
                        outfile << "[" <<filepath[i] << "] =pair with= [" << filepath[j] << "] =
Value : " << results << endl;
                        }
                }
        }
}
outfile.close();


 if( remove( "file1_c.txt" ) != 0 )
  perror( "Error deleting file" );
 else
```

```
   puts( "File successfully deleted" );


  if( remove( "file2_c.txt" ) != 0 )
   perror( "Error deleting file" );
  else
   puts( "File successfully deleted" );
return 0;
}
```

ฟังก์ชั่นหลักนี้รับค่าอาร์กุเมนท์มาหนึ่งตัวเป็นพาธของสถานที่เก็บไฟล์ทั้งหมดที่ต้องการตรวจสอบ และทำการใส่รายชื่อทั้งหมดของไฟล์ลงไปในตัวแปรแบบเวคเตอร์ตัวหนึ่ง ต่อมาก็ทำการจัดไฟล์ทั้งหมดลงในการวนที่จะจับคู่ไฟล์ที่มีทุกไฟล์และผ่านชื่อไฟล์ที่จับคู่ไว้ลงในฟังก์ชั่น Pdetect เพื่อหาค่าความเหมือนกันนำมาเขียนลงในไฟล์แสดงผล และทำไปเรื่อยๆ จนกว่าไฟล์ที่มีจะจับคู่หมด จะได้ไฟล์เอกสารแสดงผลที่บรรจุผลการตรวจสอบไฟล์ทั้งหมดมา

จากการทำงานของฟังก์ชั่น Pdetect จะมีไฟล์สำรองที่เขียนขึ้นเพื่อใช้ในการตรวจสอบค้างอยู่ จึงต้องมีการลบออกในตอนท้ายด้วย

**ซอร์สโค้ดของ Jplag**

**ฟังก์ชั่น** Lex_caller

```
void lex_caller(string a, string b)
{
        string to_e;
        string to_f;
        string path = "lexer.exe < " ;
        string filea = path+a;
        string fileb = path+b;
        char* k = &filea[0];
        char* l = &fileb[0];
        const char* x = k;
        const char* y = l;

        system (x);
        to_e = read_sav();
        system (y);
        to_f = read_sav();

        int vall = GST(to_e,to_f);
        result_keeper(a,b,vall);
};
```

**ฟังก์ชั่นนี้ใช้แปลงตัวแปรที่ใช้เก็บพาธของไฟล์ที่จะตรวจสอบให้อยู่ในรูปแบบคำสั่งบนคอมมานด์พร้อมท์ของดอส ตามด้วยการเรียก** Lexer **และเก็บ** Token **ที่ได้ลงในตัวแปรซึ่งจะผ่านไปให้ฟังก์ชั่น** GST **และ** Result_keeper **เพื่อทำการคำนวณค่าความเหมือนและเขียนผลของการตรวจสอบลงบนไฟล์เอกสารเพื่อแสดงผล**

**ฟังก์ชั่น** Result_keeper

```
void result_keeper(string filea , string fileb,int result)
{
    ofstream myfile;
    myfile.open ("jresult.txt" , ios_base::out | ios_base::app);
    myfile << " [ " << filea << " ] =pair with= [ " << fileb << " ] value = " << result << endl;
    myfile.close();
};
```

ทำการเขียนผลที่ได้จากการคำนวณด้วย Jplag ลงบนไฟล์แสดงผลชื่อ Jresult.txt

**ฟังก์ชั่น** Read_sav

```
string read_sav()// generate token to file "save.txt" and read it.
{
    string line;
    ifstream myfile ("sav.txt");
    if (myfile.is_open())
    {
        while (! myfile.eof() )
        {
        getline (myfile,line);
        cout << "the read file is " << line << endl;
        }
        myfile.close();
    }
    return line;

};
```

ทำการอ่านไฟล์โทเคน (Token) จากไฟล์ Sav.txt เข้ามาทีละบรรทัด แล้วส่งออกไปให้ฟังก์ชั่นที่เรียก

ฟังก์ชั่น GST

```
int GST(string a, string b)
{
        int ai=a.size();
        int bi=b.size();
        int count=0;

        for (int i = 0 ; i < ai ; i++ )
        {
                for (int j = 0 ; j < bi ; j++)
                {
                        char x = a[i];
                        char y = b[j];

                        if (x==y)
                        {
                                if (a[i+1]==b[j+1])
                                {
                                        count = count +1;
                                        i=i+1;
                                }
                                else
                                {
                                        count = count +1;
                                        break;
                                }
                        }
```

```
            }
        }
```

คือการตรวจสอบชุดของโทเคนเพื่อหาความเหมือนกันในชุดโปรแกรม โดยจะดูว่ามีค่าเหมือนกันยาวที่สุดเท่าไหร่ แล้วคำนวณจากความยาวทั้งหมดออกมา โดยที่ Lexer จะทำการสร้างชุดโทเคนออกมาสำหรับกรณีนี้อีกทีหนึ่ง

ฟังก์ชั่น Main

```
int main(int argc, const char* argv[])
{
        ofstream clear_rst;
        clear_rst.open ("jresult.txt" , ios_base::out | ios_base::trunc);
        clear_rst << endl;
        clear_rst.close();

        //paring(argc,argv);
////////////////////////////////////////////////////////////////////
        const char* dir_path = argv[1];

        WIN32_FIND_DATA f;
        HANDLE h = FindFirstFile(dir_path, &f);
        vector<string> filepath;

        if(h != INVALID_HANDLE_VALUE){
                do{
                        string filename = dir_path;
                        filename.erase(filename.length()-1,1);
                        filename.append(f.cFileName);
                        if(filename.find(".cpp") != filename.npos){
                        filepath.push_back(filename);
```

```
                    }
                }
                while(FindNextFile(h, &f));
        }
        else{
        fprintf(stderr, "Error opening directory\n");
        }

        for (int i = 0; i < filepath.size(); i++)
        cout << filepath[i] << endl;

        for (int i = 0; i < filepath.size(); i++)
        {
                for (int j = 0; j < filepath.size(); j++)
                {
                    if(i!=j)
                    {
                        string a = filepath[i];
                        string b = filepath[j];
                        lex_caller(a,b);
                    }
                }
        }

////////////////////////////////////////////////////////////////////////
        return 0;
}
```

ฟังก์ชั่นหลักนี้รับค่าอาร์กุเมนท์มาหนึ่งตัวเหมือนกับของ Pdetect ซึ่งรับพาธของสถานที่เก็บไฟล์ทั้งหมดที่ต้องการตรวจสอบ และทำการใส่รายชื่อทั้งหมดของไฟล์ลงไปในตัวแปรแบบเวคเตอร์ตัวหนึ่ง ต่อมาก็ทำการจัดไฟล์ทั้งหมดลงในการวนที่จะจับคู่ไฟล์ที่มีทุกไฟล์และผ่านชื่อไฟล์ที่จับคู่และเรียกด้วยฟังก์ชั่น Lex_caller เพื่อหาค่าความเหมือนกันนำมาเขียนลงในไฟล์แสดงผล และทำไปเรื่อยๆ จนกว่าไฟล์ที่มีจะจับคู่หมด จะได้ไฟล์เอกสารแสดงผลที่บรรจุผลการตรวจสอบไฟล์ทั้งหมดมา

**ความต้องการเบื้องต้น**

CPU 1.0 GHz

RAM 512 Mb

พื้นที่ว่างใน Hard Disk 15 Mb

dot NET framework 2.0

Windows XP Service Pack 2

**อินเตอร์เฟสของโปรแกรม**


ภาพที่ ก.1 : อินเตอร์เฟสของโปรแกรม

ประกอบไปด้วยคอนโทรลต่างๆ ดังนี้

- ปุ่ม Select Folder : ใช้ในการเลือกที่เก็บไฟล์ที่ต้องการตรวจสอบ เมื่อกดแล้วจะได้หน้าต่างสำหรับเลือกไฟล์


ภาพที่ ก.2 : หน้าต่างสำหรับเลือกโฟลเดอร์เป้าหมาย

และเมื่อเลือกแล้วจะมีข้อความแสดงว่าได้เลือกอะไรไว้ และจะปรากฏอยู่ในช่อง Current Path ด้วย


ภาพที่ ก.3 : แสดงโฟลเดอร์เป้าหมาย

- ช่อง Current path : แสดงพาธที่เลือกไว้
- ปุ่ม Calculate : จะแสดงออกมาเมื่อเลือกพาธแล้วเท่านั้น ใช้สำหรับสั่งให้โปรแกรมทำการคำนวณหาค่าความเหมือนกันของพาธที่กำหนด
- ช่อง Pdetect result และ Jplag result : แสดงผลที่ได้จากการคำนวณด้วยอัลกอริธึม Pdetect และ Jplag