CPE217 – Homework 9

Homework: Graph Data Structure

Homework Due Date: 3 December 2017

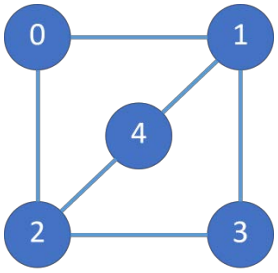
Patiwet Wuttisarnwattana, Ph.D.

Department of Computer Engineering

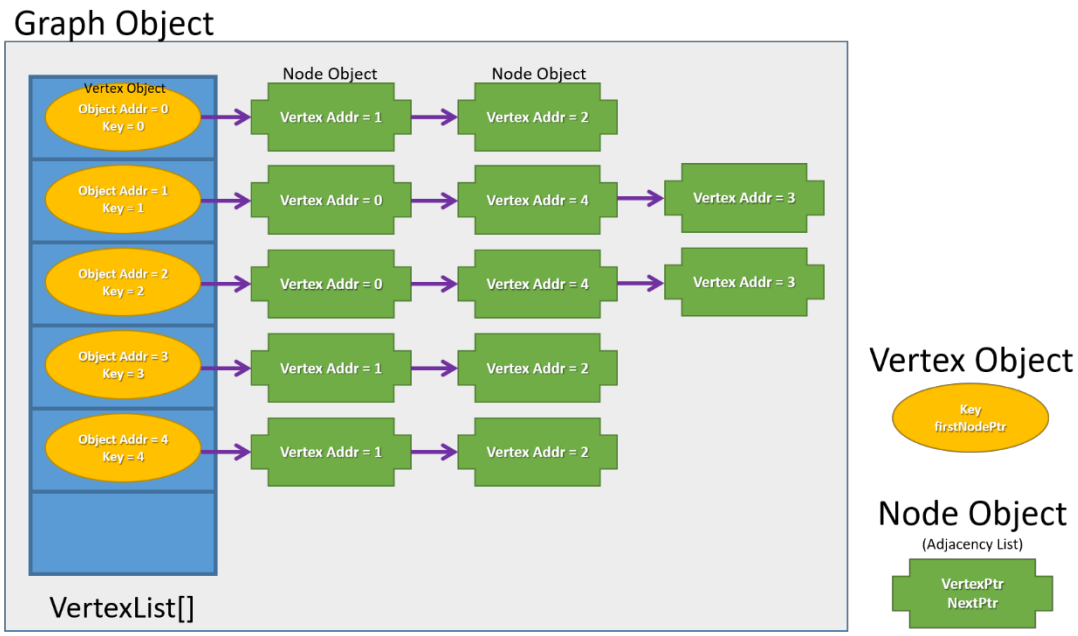
- คำชี้แจงการส่งงาน
- แต่ละกลุ่ม ควรให้ core person เป็นคนส่งงาน และในช่องข้อความต้องระบุรหัสประจำตัวนักศึกษาของทุกคนที่เป็นสมาชิกในกลุ่ม หาก core person ไม่สามารถส่งงานได้ ให้สมาชิกคนใดก็ได้ส่งงานแทน แต่ต้องบอกว่าส่งแทน core person ซึ่งก็คือใคร มีรหัสอะไร
- โค้ดของคุณต้องมีคอมเม้นต์ (comment) เพื่ออธิบายว่าโค้ดดังที่เห็นอยู่นี้ทำงานอะไร หรือ if นี้ทำตรวจหาอะไร หากกลุ่มไหนไม่มีคอมเม้นต์ในโค้ดจะไม่ได้รับการตรวจ การเขียนคอมเม้นต์ไม่ต้องเขียนแบบละเอียดยิบก็ได้ เท่าที่คุณต้องการให้ผู้ตรวจทราบก็พอ
- งานที่ส่งต้องประกอบด้วย ZIP file ของ src folder ที่สามารถกด F6 รันได้เลย หากมี compile error หรือ runtime exception งานของนักศึกษาจะไม่ได้รับการตรวจ
- สามารถส่งการบ้านช้าได้ แต่หักคะแนนวันละ 10%
- การลอกงานเพื่อนมาส่ง เป็นการทุจริตและมีความผิดทางวินัย หากตรวจพบอาจารย์อาจพิจารณาให้คะแนนการบ้านนั้นหรือคะแนนการบ้านทั้งหมดของคุณ และ/หรือ คะแนนจิตพิสัย ทั้งหมดได้ศูนย์คะแนน ซึ่งนั่นอาจเป็นปัจจัยของการตัดสินใจถอนกระบวนวิชานี้ของคุณและลงทะเบียนใหม่ในปีการศึกษาหน้า
- การลอกงานมาส่งต้องรับผิดชอบพร้อมกันทั้งกลุ่ม จะให้คนทำผิด รับผิดชอบแต่เพียงคนเดียวไม่ได้
- เพื่อนในกลุ่มที่เหลือมีหน้าที่ต้องตรวจสอบความถูกต้องและรับประกันผลงานว่าไม่ได้นำผลงานของกลุ่มอื่นมาส่ง

การบ้านนี้นักศึกษาจะได้เรียนการ implement โค้ดภาษาจาวาเพื่อสร้าง Graph Data Structure ด้วยวิธีการ List of Adjacency กล่าวคือ Graph จะมีตัวแปร vertexList ซึ่งเป็น Array of Vertex ที่จะทำหน้าที่บรรจุ Vertex (ตัว Vertex จะบรรจุข้อมูล key) นอกจากนี้ตัว Vertex เองก็ประพฤติตัวเป็น List อีกด้วย โดยให้ Vertex สามารถชี้ไปที่ Node Object โดยใน Node จะบรรจุข้อมูล ว่า Vertex ดังกล่าวเชื่อมต่อไปยัง Vertex ใด โดยกำหนดให้ Node หนึ่ง Node สามารถชี้ไปยัง Vertex ได้เพียง Vertex เดียวและ Node ใด ๆ สามารถที่จะชี้ต่อไปยัง Node อื่น ๆ ได้อีกในลักษณะของ Singly Linked List without Tail (หัวขบวนเป็น Vertex หลังจากนั้นเป็น Node)

เพื่อให้เกิดความเข้าใจในโมเดลนี้ อาจารย์ขอยกตัวอย่างแผนภาพกราฟดังต่อไปนี้



ซึ่งเมื่อแปลงเป็นโครงสร้างข้อมูลตามการบ้านนี้แล้ว กราฟดังกล่าวจะมีหน้าตาเป็นแบบนี้



จะเห็นได้ว่า Vertex ที่มี Key 0 เบื้องต้นจะให้อยู่ที่ vertexList[0] หากต้องการจะหาว่า Vertex Key 0 เชื่อมต่อกับ Vertex อะไรบ้างให้ดูสาย List of Node ที่อยู่ด้านขวาของ Vertex โดยในแต่ละ Node จะบรรจุข้อมูล Adjacent Vertex เอาไว้ ดังตัวอย่าง Vertex 0 จะเชื่อมต่อ (Connect) กับ Vertex 1 กับ Vertex 2

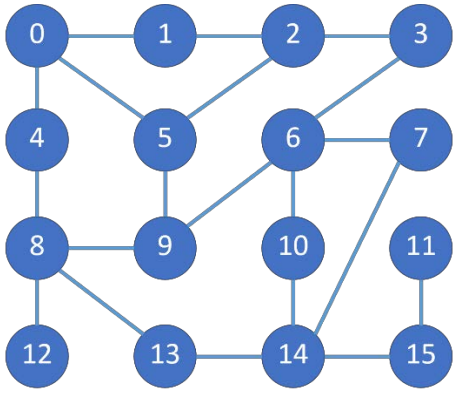
นักศึกษาจะเห็นว่าการสร้าง class Graph จะประกอบด้วย Array of Vertex (บรรจุ Vertex), class Vertex (บรรจุ Key) และ class Node (ชี้กลับไปยัง Vertex)

การบ้านนี้นักศึกษาต้องแก้ไข/เพิ่มเติมโค้ดภาษาจาวาของอาจารย์ให้สมบูรณ์เพื่อที่โปรแกรมจะทำงานได้ตามที่กำหนด นักศึกษาจะได้รับ Source code ที่ประกอบด้วย class ที่สำคัญที่คุณต้องแก้ไขเพิ่มเติม ดังต่อไปนี้

1. Class Graph ทำหน้าที่เป็น Graph Data Structure ตามวิธีการ List of Adjacency โดย class Graph มี Method ที่สำคัญดังต่อไปนี้
    a. public void addVertex(int key) ทำหน้าที่ บรรจุ key ลงใน Object Vertex แล้วนำ Vertex ดังกล่าวไปบรรจุใน Array vertexList ในตำแหน่งที่ถูกต้อง (เบื้องต้นให้ใส่ตำแหน่งเดียวกันกับ Key)
    b. public void addEdge(int source, int destination) ทำหน้าที่ เชื่อมต่อระหว่าง Vertex ที่มี key = source และ Vertex ที่มี key = destination ในทางปฏิบัติก็คือ สร้าง Node ขึ้นมาพร้อมกับบรรจุ reference ของ Vertex key = destination แล้วนำไปต่อ (append) กับ Vertex key = source
    c. public void DFS() ทำหน้าที่ เดินทางจาก Vertex แรกสุดของ Adjacency List แล้วไปต่อในทุก ๆ Vertex แบบ Depth First Search ตามที่เรียนในห้อง ซึ่งในการบ้านนี้กระบวนการ DFS จะเป็นส่วนหนึ่งของอัลกอริทึม Connected Component Labeling
    d. public void Explore(Vertex v) ทำหน้าที่ ตรวจสอบในทุก ๆ Adjacent Vertex ของ Vertex v นั้นมี Vertex ที่ถูก visit แล้วหรือยัง ถ้ายังก็ให้ Explore ต่อไปแบบ Recursive ตามที่เรียนในห้อง กระบวนการ Explore นี้ก็เป็นส่วนหนึ่งของอัลกอริทึม Connected Component Labeling
    e. public void BFS(int s) ทำหน้าที่ เดินทางจาก Vertex key = s ไปยังทุก ๆ Vertex ที่ติดกัน โดยเดินทางไปแบบ Breadth First Search ตามที่เรียนในห้อง ซึ่งในการบ้านนี้กระบวนการ BFS จะเป็นส่วนหนึ่งของอัลกอริทึม Shortest Flight Segment Route
    f. public Stack getShortestPathList(int s_key, int u_key) ทำหน้าที่สร้าง LIFO List ที่บรรจุข้อมูล Vertex ทั้งหมดที่อยู่บนเส้นทางที่สั้นที่สุดจาก Vertex key = s_key ถึง Vertex key = u_key อัลกอริทึม getShortestPathList นี้เป็นส่วนหนึ่งของอัลกอริทึม Shortest Flight Segment Route
    g. public void printShortestPart(int s, int u) ทำหน้าที่แสดงค่า Key ของ Vertex ตั้งแต่ S ถึง U ฟังก์ชันนี้ทำหน้าที่เรียกฟังก์ชัน BFS และ getShortestPathList และทำหน้าที่แสดงค่าออกทาง Console โดยอัตโนมัติ อาจารย์ Implement ไว้ให้แล้วคุณไม่ต้องแก้ไขอะไรเพิ่ม
2. Class HashGraph จะทำหน้าที่ได้คล้ายกับ class Graph ตามที่อธิบายก่อนหน้า นอกจากนี้ อาจารย์ยังสั่งให้ class HashGraph สืบทอดคุณสมบัติ (extends) ของ class Graph มาทั้งหมดอีกด้วย ทั้งนี้ สิ่งที่แตกต่างกันออกไป คือ class HashGraph จะรับ Key เข้ามาเป็น String แทนที่จะเป็นจำนวนเต็มขนาดเล็ก เหมือน class Graph เพราะฉะนั้น การ Add Vertex ลงในตำแหน่งของ Array vertexList ก็จะไม่ตรงกับค่าของ Key (String) แต่จะตรงกับค่าของ Key ที่ถูกแฮช (Hash) แล้วด้วยฟังก์ชัน PolyHash ตามที่เรียนในห้อง และ ถ้าหากค่าของ Key นั้นมีข้อมูลอยู่ก่อนแล้ว (เพราะเกิดการชน) ก็ให้ Hash ใหม่ด้วยวิธีการของ Quadratic Probing ตามที่เรียนในห้อง จนกว่าจะพบที่ว่างที่จะสามารถ Add Vertex ได้
3. สำหรับรายละเอียดของ Class อื่น ๆ หรือฟังก์ชันอื่น ๆ ขอให้นักศึกษา ศึกษาเองตามตัวอย่างด้านล่าง

ตัวอย่างการทำงานที่ 1

กำหนดให้โครงสร้างข้อมูลกราฟ คือแผนภาพดังต่อไปนี้ โดยมี Vertex ทั้งสิ้น 16 Vertices และมี Edge ทั้งสิ้น 20 Edges



เมื่อนักศึกษาแก้โค้ดอาจารย์ได้เสร็จสิ้นแล้ว โค้ดดังต่อไปนี้ควรที่จะสามารถสร้าง Graph ได้ตามกำหนด

```
public static void main(String[] args) {
        Graph graph = new Graph(32);
        for (int i=0; i<16; i++)
            graph.addVertex(i);

        graph.addEdge(0, 1);        graph.addEdge(0, 5);        graph.addEdge(0, 4);        graph.addEdge(1, 2);
        graph.addEdge(2, 5);        graph.addEdge(2, 3);        graph.addEdge(3, 6);        graph.addEdge(4, 8);
        graph.addEdge(5, 9);        graph.addEdge(6, 7);        graph.addEdge(6, 10);       graph.addEdge(6, 9);
        graph.addEdge(7, 14);       graph.addEdge(8, 9);        graph.addEdge(8, 13);       graph.addEdge(8, 12);
        graph.addEdge(10, 14);      graph.addEdge(11, 15);      graph.addEdge(13, 14);      graph.addEdge(14, 15);
}
```

เมื่อเราต้องการเช็คว่า Vertex key = 0 มี Vertex ที่เป็นเพื่อนบ้านกัน (มี Edge เชื่อมหากัน) มีอะไรบ้าง นักศึกษาสามารถที่จะเรียกคำสั่ง

graph.vertexList[0].showList();
ผลลัพธ์ที่ได้คือ
Vertex 0 connected to the following vertices: 1, 5, 4,

คุณสามารถเช็ค Vertex อื่น ๆ ได้อีกว่าเชื่อมต่อกันได้อย่างถูกต้องหรือไม่

```
graph.vertexList[1].showList();
graph.vertexList[5].showList();
graph.vertexList[14].showList();
graph.vertexList[11].showList();
```

ผลลัพธ์ที่ได้คือ

```
Vertex 1 connected to the following vertices: 0, 2,
Vertex 5 connected to the following vertices: 0, 2, 9,
Vertex 14 connected to the following vertices: 7, 10, 13, 15,
Vertex 11 connected to the following vertices: 15,
```

ต่อไปเป็นการทดสอบ Depth First Search คุณสามารถเรียกใช้คำสั่งคือ

```
graph.DFS();
```

ผลลัพธ์ที่ได้คือ

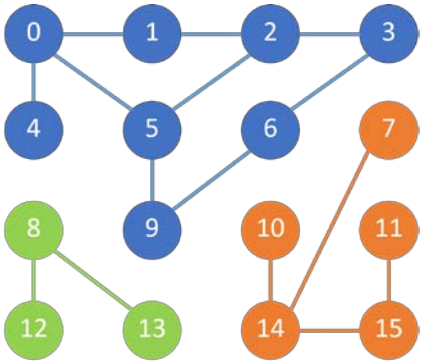
```
0/1 -> 1/1 -> 2/1 -> 5/1 -> 9/1 -> 6/1 -> 3/1 -> 7/1 -> 14/1 -> 10/1 -> 13/1 -> 8/1 -> 4/1 -> 12/1 -> 15/1 -> 11/1 ->
```

ผลลัพธ์จะรายงานสองตัวเลขต่อหนึ่ง Vertex คั่นโดย / (slash) โดยตัวเลขแรกแสดงถึงค่า Key ของ Vertex ส่วนตัวเลขที่สองคือ Connected Component Label การที่ทุกค่าให้ Connected Component Label เป็นค่าเดียวกันหมดแปลว่า ทุก ๆ Vertex เชื่อมต่อหากัน/มีเส้นทางหากันในทุก ๆ Vertex

การเดินทางจาก Vertex แรกไปยัง Vertex สุดท้ายสามารถอ่านได้จากตัวเลขตัวแรก ดังตัวอย่างคือ 0, 1, 2, 5, 9, 6, 3, 7, 14, 10, 13, 8, 4, 12, 15, 11

ตัวอย่างการทำงานที่ 2

กำหนดให้โครงสร้างข้อมูลกราฟ คือแผนภาพดังต่อไปนี้ โดยมี Vertex ทั้งสิ้น 16 Vertices และมี Edge ทั้งสิ้น 15 Edges



โค้ดดังต่อไปนี้ควรที่จะสามารถสร้าง Graph ได้ตามกำหนด

```
public static void main(String[] args) {
        Graph graph = new Graph(32);
        for (int i=0; i<16; i++)
            graph.addVertex(i);

        graph.addEdge(0, 1);        graph.addEdge(0, 5);        graph.addEdge(0, 4);        graph.addEdge(1, 2);
        graph.addEdge(2, 5);        graph.addEdge(2, 3);        graph.addEdge(3, 6);        graph.addEdge(5, 9);
        graph.addEdge(6, 9);        graph.addEdge(7, 14);       graph.addEdge(8, 13);       graph.addEdge(8, 12);
        graph.addEdge(10, 14);      graph.addEdge(11, 15);      graph.addEdge(14, 15);
}
```

และเมื่อประยุกต์ Depth First Search เพื่อนับจำนวน Connected Component ตามที่เรียนในห้อง คุณสามารถที่เรียกใช้คำสั่งได้ดังนี้

```
graph.DFS();
System.out.println("\nNumber of connected component = " + (graph.cc-1));
```

ผลลัพธ์ที่ได้คือ

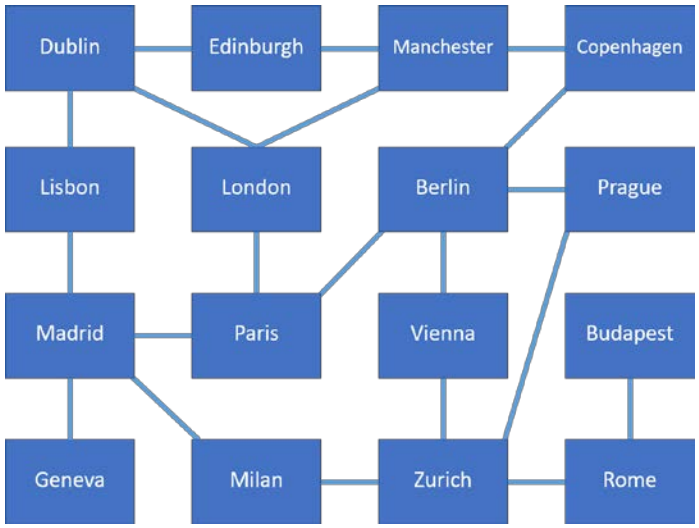
```
0/1 -> 1/1 -> 2/1 -> 5/1 -> 9/1 -> 6/1 -> 3/1 -> 4/1 -> 7/2 -> 14/2 -> 10/2 -> 15/2 -> 11/2 -> 8/3 -> 13/3 -> 12/3 ->
Number of connected component = 3
```

ผลลัพธ์จะแสดงให้เห็นว่า Vertex แต่ละอันนั้นอยู่ในกลุ่มไหน ซึ่งจากรูปจะมีอยู่สามกลุ่มและโปรแกรมก็นับจำนวนกลุ่มให้ได้อย่างถูกต้องในผลลัพธ์ สังเกตด้วยว่าในการเดินทางในกลุ่มเดียวกัน จะเป็นการเดินทางแบบ DFS โดยเริ่มจาก Vertex แรกสุดที่อยู่ในลิสท์ (Vertex key 0; ทำไม?)

คำถามต่อไปคือ ทำไมในโค้ดอาจารย์ต้องลบหนึ่งออกจาก graph.cc ด้วย เพื่อที่จำนับจำนวน Connected Component ให้ถูกต้อง ให้นักศึกษาคิดและนำคำตอบไปตอบในข้อสอบ

ตัวอย่างการทำงานที่ 3

กำหนดให้โครงสร้างข้อมูลกราฟ คือแผนภาพดังต่อไปนี้ โดยมี Vertex ทั้งสิ้น 16 Vertices และมี Edge ทั้งสิ้น 20 Edges และ Key ที่บรรจุไว้ใน Vertex แต่ละตัวนั้น กำหนดให้เป็น String ของชื่อเมืองหลวง (Capital Cities)



จงประยุกต์อัลกอริทึม Breadth First Search ตามที่เรียนในห้องเพื่อหาเส้นทางที่สั้นที่สุดระหว่างเมืองสองเมืองใด ๆ โดยกำหนดให้การแก้ปัญหานี้ ให้ใช้ class HashGraph ที่มีแฮชฟังก์ชันที่ทำหน้าที่แปลงชื่อเมืองไปเป็น Array Index ให้ ซึ่งถ้า Array Index ที่แฮชได้เกิดการชน (Collision) โปรแกรมก็จะแก้ปัญหาการชนด้วย Quadratic Probing ตามที่เรียนในห้อง อาจารย์กำหนดให้คุณใช้ Parameters ดังต่อไปนี้เพื่อสร้าง Polynomial Hashing ตามที่เรียนในห้อง โดยมี p = 101111, x = 101, m = 32

โค้ดดังต่อไปนี้ควรที่จะสามารถสร้าง Graph ได้ตามกำหนด

```
public static void main(String[] args) {
        long p = 101111; // Big Prime (Hash key1)
        long x = 101; // Small number (Hash key2)
        HashGraph graph = new HashGraph(32, p, x);
        String[] cities = new String[]["Dublin", "Edinburgh", "Manchester",
            "Copenhagen", "Lisbon", "London", "Berlin", "Prague", "Madrid",
            "Paris", "Vienna", "Budapest", "Geneva", "Milan", "Zurich", "Rome"];
        for (int i=0; i<16; i++){
            graph.addVertex(cities[i]);
        }
```

```
        graph.addEdge("Dublin", "Edinburgh");          graph.addEdge("Dublin", "London");
        graph.addEdge("Dublin", "Lisbon");          graph.addEdge("Edinburgh", "Manchester");
        graph.addEdge("Manchester", "London");          graph.addEdge("Manchester", "Copenhagen");
        graph.addEdge("Copenhagen", "Berlin");          graph.addEdge("Lisbon", "Madrid");
        graph.addEdge("London", "Paris");          graph.addEdge("Berlin", "Prague");
        graph.addEdge("Berlin", "Vienna");          graph.addEdge("Berlin", "Paris");
        graph.addEdge("Prague", "Zurich");          graph.addEdge("Madrid", "Paris");
        graph.addEdge("Madrid", "Milan");          graph.addEdge("Madrid", "Geneva");
        graph.addEdge("Vienna", "Zurich");          graph.addEdge("Budapest", "Rome");
        graph.addEdge("Milan", "Zurich");          graph.addEdge("Zurich", "Rome");
}
```

เมื่อเราต้องการเช็คว่าเมือง Paris, Zurich, Geneva ติดต่อกับเมืองใดบ้าง นักศึกษาสามารถที่จะเรียกคำสั่งดังต่อไปนี้

```
graph.vertexList[graph.hashMapWithQuadraticProbing("Paris")].showList();
graph.vertexList[graph.hashMapWithQuadraticProbing("Zurich")].showList();
graph.vertexList[graph.hashMapWithQuadraticProbing("Geneva")].showList();
```

ผลลัพธ์ที่ได้คือ

```
Vertex Paris connected to the following vertices: London, Berlin, Madrid,
Vertex Zurich connected to the following vertices: Prague, Vienna, Milan, Rome,
Vertex Geneva connected to the following vertices: Madrid,
```

เมื่อเราต้องการค้นหาเส้นทางว่า เส้นทางใดที่สั้นที่สุดระหว่างเมือง London ไปยัง Budapest และ เมือง Berlin ไป Dublin (ว่าต้องผ่านเมืองไหนบ้าง) นักศึกษาสามารถที่จะเรียกคำสั่งดังต่อไปนี้

```
graph.printShortestPart("London", "Budapest");
graph.printShortestPart("Berlin", "Dublin");
```

ผลลัพธ์ที่ได้คือ

```
London -> Paris -> Berlin -> Prague -> Zurich -> Rome -> Budapest ->
Berlin -> Paris -> London -> Dublin ->
```

ซึ่งเมื่อตรวจสอบกับแผนภาพด้านบนก็จะพบว่า เส้นทางที่แสดงนี้เป็นเส้นทางที่สั้นที่สุดแล้ว

หากการบ้านนี้นักศึกษาไม่รู้จะไปยังไง ไม่เข้าใจเลย หรือเสียเวลาการทำการบ้านมากเกินไป ขอให้นักศึกษาเข้ามาปรึกษากับอาจารย์เป็นการด่วน เพื่อที่จะไม่เสียเวลาในการอ่านเตรียมสอบวิชาอื่นต่อไป