CPE217 – Homework 9

Homework: Hash Graph Data Structure

Homework Due Date: 30 November 2019

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

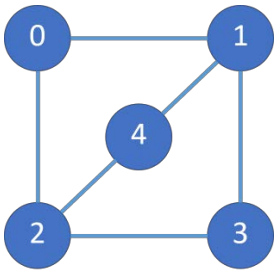
การบ้านนี้นักศึกษาจะได้เรียนการ implement โค้ดภาษาจาวาเพื่อสร้าง Graph Data Structure ด้วยวิธีการ Adjacency List กล่าวคือ Graph จะมีตัวแปรสำคัญสองตัว คือ vertexList และ adjacencyList ซึ่งตัวแปรทั้งสองจะมีสภาพเป็น Array มีขนาดเท่ากัน

ตัวแปรแรกมีชื่อว่า vertexList (ซึ่งเป็นชนิด Array of Vertices) มีหน้าที่คือ บรรจุข้อมูลจริงซึ่งก็คือ Vertex ซึ่งตัว Vertex นั้นจะทำหน้าที่บรรจุข้อมูลสำคัญต่าง ๆ เช่น key, ccNum, dist, visited ซึ่งจะเป็นคุณสมบัติ (Attribute) ของกราฟที่สำคัญและจะถูกใช้ในการประมวลผลทางกราฟ เช่น BFS และ DFS ต่อไป

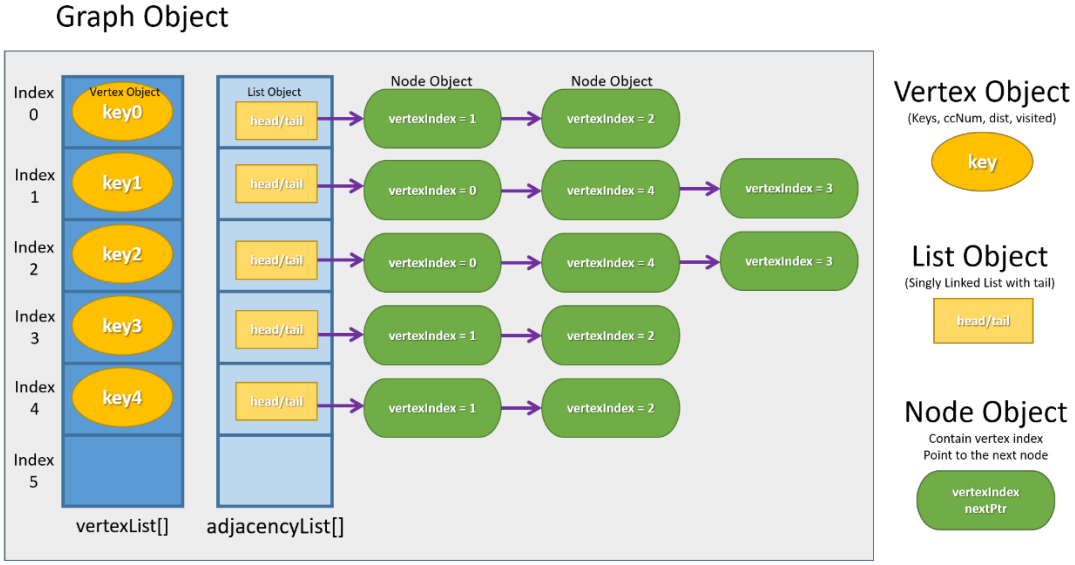
ส่วนตัวแปรที่สองนั้นมีชื่อว่า adjacencyList (ซึ่งเป็นตัวแปรชนิด Array of Lists) มีหน้าที่คือ ระบุว่า Vertex ดังกล่าวนั้นประชิด (Adjacent) กับ Vertex ใดบ้าง โดยตัวแปรนี้จะบรรจุ List ของ Node โดยตัวแปรชนิด Node นั้นจะบรรจุข้อมูลสำคัญคือ vertexIndex (ชนิด integer) หน้าที่ของตัวแปรนี้คือทำการระบุตำแหน่งในของ Array ที่บรรจุ Vertex ที่อยู่ติดกัน

Array ที่ชื่อ vertexList และ adjacencyList ถูกออกแบบมาให้มีขนาดเท่ากัน ทำให้ตำแหน่ง Index เดียวกัน จะอ้างถึงข้อมูล Vertex ตัวเดียวกัน

เพื่อให้เกิดความเข้าใจในโมเดลนี้ อาจารย์ขอยกตัวอย่างแผนภาพกราฟดังต่อไปนี้



ซึ่งเมื่อแปลงเป็นโครงสร้างข้อมูลตามการบ้านนี้แล้ว กราฟดังกล่าวจะมีหน้าตาเป็นแบบนี้



จะเห็นได้ว่า Vertex ที่มี Key 0 เบื้องต้นจะให้อยู่ที่ vertexList[0] หากต้องการจะหาว่า Vertex Key 0 นี้เชื่อมต่อกับ Vertex อะไรบ้างให้ดูใน List ที่ตำแหน่ง adjacencyList ที่มี Index ตรงกันคือ adjacencyList[0] ซึ่งจะปรากฏสาย Node list (ขบวนสีเขียวที่อยู่ด้านขวา) โดยในแต่ละ Node จะบรรจุข้อมูล Index ของ Vertex ที่เชื่อมต่อกันเอาไว้ (ตามที่ระบุใน vertexList) ดังตัวอย่าง Vertex 0 ที่จะเชื่อมต่อกับ Vertex 1 และ Vertex 2 ก็จะปรากฏว่า List ที่ตำแหน่ง adjacencyList[0] จะมีขบวน Node ที่บรรจุ Index คือ 1 กับ 2

อีกตัวอย่างหนึ่ง ก็คือ Vertex 1 ที่เชื่อมต่อกับ Vertex 0, Vertex 4, Vertex 3 ก็จะมีขบวนของ Node ที่บรรจุ Index 0, 4, 3 ตามลำดับ

นักศึกษาจะสามารถสังเกตได้ว่า หาก Vertex 0 เชื่อมต่อกับ Vertex 1 ตามที่ระบุไว้ใน adjacencyList[0] แล้ว Vertex 1 ก็จะเชื่อมต่อกับ Vertex 0 ตามที่ระบุไว้ใน adjacencyList[1] ด้วยเช่นกัน ความสัมพันธ์นี้จะไปอย่างสมมาตร ดังนั้นเวลานักศึกษา pushBack Node(1) ไปยัง List[0] แล้ว ก็อย่าลืม pushBack Node(0) ไปยัง List[1] ด้วย สำหรับขั้นตอนการสร้างกราฟ (อยู่ในฟังก์ชัน addEdge)

การบ้านนี้นักศึกษาต้องแก้ไข/เพิ่มเติมโค้ดภาษาจาวาของอาจารย์ให้สมบูรณ์เพื่อที่โปรแกรมจะทำงานได้ตามที่กำหนด โดยนักศึกษาจะได้รับ Source code ที่ประกอบด้วย class ที่สำคัญ ที่คุณต้องแก้ไขเพิ่มเติมดังต่อไปนี้

1. Class Graph ทำหน้าที่เป็น Graph Data Structure ตามวิธีการ List of Adjacency โดยมี Method ที่สำคัญดังต่อไปนี้
    a. public void addVertex(int key) ทำหน้าที่ บรรจุ Vertex object ลงใน Array vertexList และสร้าง List Object ลงใน Array adjacencyList ในตำแหน่งที่ถูกต้อง (เบื้องต้นให้ใส่ตำแหน่งเดียวกันกับ Key)
    b. public void addEdge(int u, int v) ทำหน้าที่ เชื่อมต่อระหว่าง Vertex ที่มี key = u และ Vertex ที่มี key = v สำหรับการเชื่อมต่อ Vertex u ไปยัง Vertex v นั้นให้สร้าง Node object ที่บรรจุ key v แล้วนำไปบรรจุลงใน List ของ Vertex u และ เนื่องจากกราฟในวิชานี้เป็นโมเดลที่สมมาตร นักศึกษาอย่างลืมเชื่อม Vertex v กลับไปยัง Vertex u ด้วยนะครับ
    c. public void DFS() ทำหน้าที่ สำรวจโดยเริ่มจาก Vertex แรกสุดของ List แล้วสำรวจลึกลงไปต่อยัง Vertex อื่น (ที่เชื่อมต่อกัน) แบบ Depth First Search ตามที่เรียนในห้อง นอกจากนี้ Vertex ที่เชื่อมต่อกันนั้นจะถูกจัดกลุ่มให้อยู่ใน Connected Component (CC number) ที่มีเลขเดียวกันอีกด้วย
    d. public void Explore(Vertex v) ทำหน้าที่ สำรวจในทุก ๆ Vertex ที่มีเส้นเชื่อมต่อกันกับ Vertex v ว่า ถูก visit แล้วหรือยัง ถ้ายังก็ให้สำรวจต่อไปแบบ Recursive ตามที่เรียนในห้อง
    e. public void BFS(int s) ทำหน้าที่ เดินทางจาก Vertex (key = s) ไปยังทุก ๆ Vertex ที่มีเส้นเชื่อมต่อกัน โดยเดินทางไปแบบ Breadth First Search ตามที่เรียนในห้อง นอกจากนี้ฟังก์ชัน BFS จะสามารถคำนวณระยะทางของ Vertex ที่อยู่ห่างออกไปนับจาก Vertex s ด้วยตัวแปร dist พร้อมกับบอกเส้นทางผ่านตัวแปร prev อีกด้วย
    f. public Stack getShortestPathList(Vertex S, Vertex U) ทำหน้าที่สร้างเส้นทางที่สั้นที่สุด (ในรูปแบบของ LIFO List) จาก Vertex S ถึง Vertex U ซึ่งเส้นทางดังกล่าวต้องคำนวณจากฟังก์ชัน BFS เรียบร้อยแล้ว ฟังก์ชันนี้ อาจารย์สร้างให้สำเร็จแล้ว ไม่ต้องแก้ไข ใด ๆ

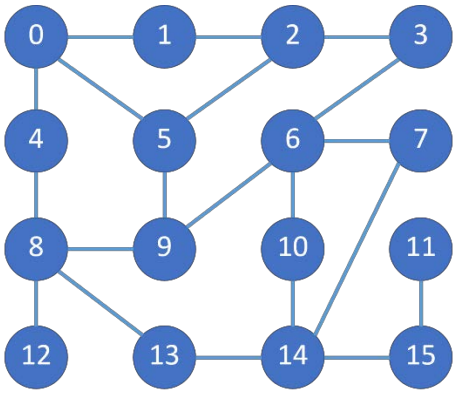
g. public void printShortestPart(String s_str, String u_str) ทำหน้าที่เรียกใช้ฟังก์ชัน BFS และ getShortestPathList ให้เหมาะสม นักศึกษาต้องเพิ่มเติม/แก้ไขโค้ดอาจารย์ให้ผลลัพธ์ที่ได้แสดงเส้นทางที่เส้นที่สุดจากชื่อเมืองที่ระบุใน s_str ไปยังเมือง u_str ให้ถูกต้อง

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 แทนที่จะเป็นจำนวนเต็ม 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 ตามที่เรียนในห้องจนกว่าจะพบที่ว่างที่จะสามารถ 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



เมื่อนักศึกษาแก้โค้ดตั้งต้นอาจารย์ได้เสร็จสิ้นแล้ว โค้ดดังต่อไปนี้ ควรที่จะสามารถสร้าง 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 เชื่อมต่อหากัน) มีอะไรบ้าง นักศึกษาสามารถที่จะเรียกคำสั่ง

```
graph.showList(0);
```

ผลลัพธ์ที่ได้คือ

```
Vertex 0 connected to the following vertices: 1, 5, 4,
```

คุณสามารถเช็ค Vertex อื่น ๆ ได้อีกว่าเชื่อมต่อกันได้อย่างถูกต้องหรือไม่

```
graph.showList(1);
graph.showList(5);
graph.showList(14);
graph.showList(11);
```

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

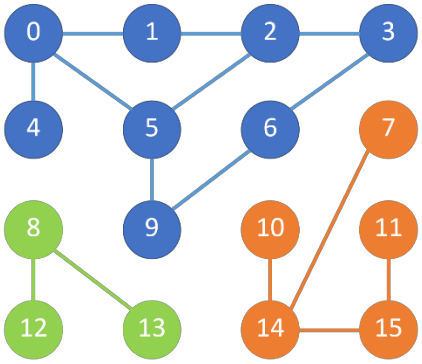
```
0/1 -> 1/1 -> 2/1 -> 5/1 -> 9/1 -> 6/1 -> 3/1 -> 7/1 -> 14/1 -> 10/1 -> 13/1 -> 8/1 -> 4/1 -> 12/1 -> 15/1 -> 11/1 ->
```

ผลลัพธ์จะรายงานสองตัวเลขต่อหนึ่ง Vertex คั่นโดย / (slash) โดยตัวเลขแรก แสดงถึงค่า Key ของ Vertex ส่วนตัวเลขที่สองคือ Connected Component Number การที่ทุกค่าให้ Connected Component Number เป็นค่าเดียวกันหมดแปลว่า ทุก ๆ Vertex เชื่อมต่อหากันและมีเส้นทางไปมาหากันในทุก ๆ Vertex

การเดินทางจาก Vertex แรกไปยัง Vertex สุดท้ายสามารถอ่านได้จากตัวเลขตัวหน้า 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

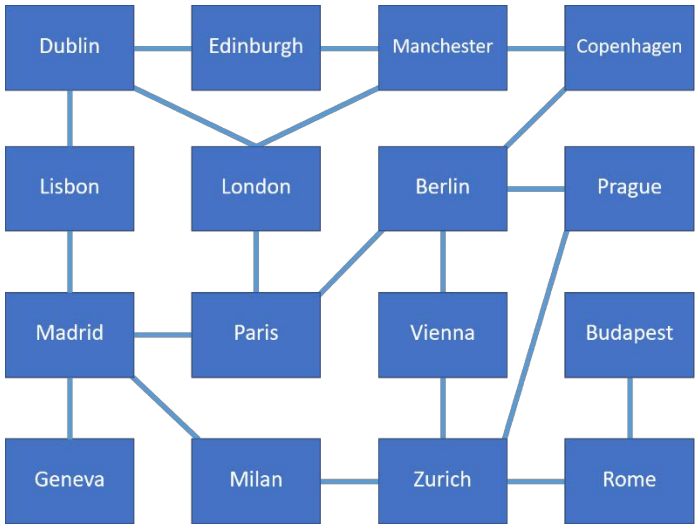
```
0/1 -> 1/1 -> 2/1 -> 5/1 -> 9/1 -> 6/1 -> 3/1 -> 4/1 -> 7/2 -> 14/2 -> 10/2 -> 15/2 -> 11/2 -> 8/3 -> 13/3 -> 12/3 ->
Number of connected component = 3
```

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

คำถามต่อไปคือ ทำไมในโค้ดอาจารย์ต้องลบหนึ่งออกจาก graph.cc ด้วย เพื่อที่จะนับจำนวน Connected Component ได้ถูกต้อง 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.showList(graph.hashCode("Paris"));
graph.showList(graph.hashCode("Zurich"));
graph.showList(graph.hashCode("Geneva"));
```

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 เส้นทางที่สั้นที่สุดระหว่างเมือง London ไปยัง Budapest และ เมือง Berlin ไป Dublin (ต้องผ่านเมืองไหนบ้าง) 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