

CPE217 – Homework 9

Homework: Hash Graph Applications

Homework Due Date: 23 October 2021

Patiwet Wuttisarnwattana, Ph.D.

Department of Computer Engineering

- คำชี้แจงการส่งงาน
- การบ้านนี้มีหัวข้อให้เรียนรู้ 3 หัวข้อ คือการทำ Hash Table, การใช้งาน Built-in Java Collections และการสร้าง Graph
- ขอให้ นศ ทำการบ้านนี้ตามหัวข้อตามลำดับ คำอธิบายอยู่ในหน้าถัดไปครับ
- เมื่อ Core Person ส่งคำตอบแล้ว ให้ Core Person เข้าโมดูล Assignment (Link สีแดง) และใส่รหัสของเพื่อนในกลุ่มลงใน ช่องคำตอบ
- TA จะตรวจคำตอบในโมดูล Quiz และนำคะแนนมาลงในโมดูล Assignment เพื่อให้ทุกคนในกลุ่มได้คะแนนเท่ากันครับ

นศ ที่จะส่งคำตอบ ท่านต้องให้คำมั่นปฎิญาณต่อคำพูดดังต่อไปนี้ หากไม่สามารถทำได้ ท่านจะไม่มีสิทธิ์ส่งงาน

- ข้าพเจ้าและเพื่อนในกลุ่มเข้าใจและตระหนักดีว่า ในการทำการบ้านนี้ ข้าพเจ้าและเพื่อนในกลุ่มจะช่วยกันทำงานนี้ให้เสร็จสิ้นเอง โดยไม่ปรึกษาหรือแบ่งปันข้อมูลกับกลุ่มอื่น ๆ หรือบุคคลภายนอก
- หากข้าพเจ้าเป็นรุ่นพี่ที่กลับมาเรียนวิชานี้อีกครั้ง ข้าพเจ้าตระหนักดีว่า ข้าพเจ้าจะทำงานให้เสร็จสิ้นเองอีกครั้ง โดยไม่ดูคำตอบของปีก่อน ๆ
- หากข้าพเจ้าไม่สามารถปฏิบัติตามคำมั่นนี้ได้ ข้าพเจ้ายินดีที่จะยอมรับคะแนน ศูนย์คะแนน ในทุก ๆ การบ้านโดยไม่ได้แย้ง

ในการบ้านนี้ นักศึกษาจะได้ศึกษาและเขียนโปรแกรมในหัวข้อสำคัญ 3 หัวข้อสุดท้ายก่อนที่เราจะปิดคอร์สจากกันไปครับ (เศร้า) หัวข้อทั้ง 3 มี ดังต่อไปนี้ครับ

1. [หัวข้อที่ 1] นักศึกษาจะได้มีโอกาสสร้าง Hash Table (ตารางแฮช) เพื่อซึ่งตามทฤษฎีแล้วสามารถทำให้การ Insert, Delete, Find ไปได้ถึง $O(1)$ เลยทีเดียว ถ้าหากว่าออกแบบกันดี ๆ
 - อย่างไรก็ตาม ในการบ้านนี้ นักศึกษาจะได้สร้าง Hash Function ที่ชื่อว่า Polynomial Hashing ที่จะแปลง String ให้กลายเป็นตัวเลขขนาดใหญ่ ผ่านพหุนามดีกรี p กับ x ตามที่เรียนในห้อง
 - หลังจากนั้น นศ จะต้องทำ Cardinality Fix เพื่อปรับตัวเลขดังกล่าวมาให้เป็นตำแหน่ง (หรือ Index) ของ Array (Hash Table) เพื่อบรรจุข้อมูล
 - ถ้าหากว่า ตำแหน่งนั้นมีข้อมูลอื่นอยู่ก่อนแล้ว นศ ก็จะทำให้การเข้าไปดูตำแหน่งถัดไปตามหลักการของ Quadratic Probing เพื่อในที่สุดแล้ว จะบรรจุข้อมูลนั้นลงไปตารางได้
 - เมื่อ นศ อ่านมาถึงจุดนี้แล้ว อ. ขอแนะนำให้นศ ไปทำการบ้านใน E-Learning ในโจทย์ข้อที่ 1 ได้เลยครับ ยังไม่ต้องอ่านต่อไปนะครับ เดียวง เมื่อทำโจทย์ข้อ 1 เสร็จแล้ว ก็มาอ่านหัวข้อที่ 2 ต่อเลยครับ
2. [หัวข้อที่ 2] หลังจากที่ นักศึกษาจะต้องสร้างคลาส LinkedList, Queue, Stack เองมาตลอดภาคการศึกษานี้ ผลลัพธ์อาจจะรันได้บ้างไม่ได้บ้าง Bugs อาจจะเยอะบ้างน้อยบ้าง อย่างไรก็ตาม นักศึกษารู้หรือไม่ว่าภาษาจาวา (และภาษาสมัยใหม่อื่น ๆ) นั้น มี Built-in class เช่น LinkedList, Queue, Stack ให้ใช้งานได้โดยไม่ต้องเขียนโปรแกรมเองนะครับ โดยโครงสร้างข้อมูลเหล่านี้สามารถบรรจุข้อมูลที่เป็นคลาสอะไรก็ได้ ได้หมดเลย ในโลกความเป็นจริงเราจะไม่พัฒนาโครงสร้างข้อมูลพวกนี้เองครับ เราจะใช้ Built-in Data Structures เหล่านี้จากจาวาเลย นศ ก็แค่ import library ที่จำเป็นเข้ามาก็จบละ (เช่น `import java.util.LinkedList;` เป็นต้น)
 - โจทย์ปัญหาข้อที่ 2 นี้ นศ จะได้ฝึกเรียนรู้การเรียกใช้งานคลาสทั้งสามนี้ด้วยตัวเองครับ
 - โครงสร้างข้อมูลทั้งสามที่เป็น Built-in class ในภาษาจาวานี้ สามารถที่จะบรรจุวัตถุใด ๆ เข้าไปก็ได้ เราอาจต้องแจ้งจาวาเพิ่มเติมหน่อยว่าวัตถุที่เราจะใส่เข้าไปเป็นชนิดไหน ผ่านสัญลักษณ์ `<>` (ภาษาอังกฤษเรียกว่า angle bracket) เช่นถ้าเราจะบรรจุ Vertex เข้าไปใน LinkedList ก็จะสามารถตัวแปรว่า `LinkedList<Vertex> list = new LinkedList();` หรือถ้าจะบรรจุตัวเลขจำนวนเต็มเข้าไปใน LinkedList ก็จะสามารถว่า `LinkedList<Integer> list = new LinkedList();` รายละเอียดวิธีการเรียกใช้งาน ขอให้ นศ ศึกษาเองได้ที่
 - <https://www.geeksforgeeks.org/linked-list-in-java/>
 - <https://www.geeksforgeeks.org/how-to-iterate-linkedlist-in-java/>
 - <https://www.geeksforgeeks.org/queue-interface-java/>
 - <https://www.geeksforgeeks.org/stack-class-in-java/>
 - และที่อื่น ๆ อีก โดยอาจารย์ขอให้นักศึกษาค้นคว้าด้วยตัวเองครับ นี่เป็นทักษะที่สำคัญมาก ๆ
 - เมื่อ นศ อ่านมาถึงจุดนี้แล้ว ก็ขอให้เข้าไปทำการบ้านใน E-Learning ในโจทย์ข้อที่ 2 ได้เลยครับ ยังไม่ต้องอ่านต่อไปนะครับ เดียวง เมื่อทำโจทย์ข้อ 2 เสร็จแล้ว ก็มาอ่านหัวข้อที่ 3 ต่อเลยครับ

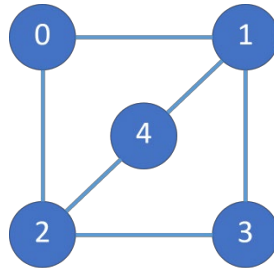
หัวข้อสุดท้าย จะเป็นการอธิบายเพื่อให้นักศึกษาเรียนรู้วิธีการ implement โค้ดภาษาจาวาเพื่อสร้าง Graph Data Structure ด้วยวิธีการ Adjacency List กล่าวคือโครงสร้างข้อมูลชนิดกราฟจะมีตัวแปรสำคัญสองตัว คือ vertexList และ adjacencyList ซึ่งตัวแปรทั้งสองจะมีสภาพเป็น Array และมีขนาดเท่ากัน

ตัวแปรแรกมีชื่อว่า vertexList (ซึ่งเป็นชนิด array of Vertex หรือ Vertex[]) โดยแต่ละ Vertex ที่บรรจุใน Array นี้ จะทำหน้าที่ บรรจุข้อมูลสำคัญต่าง ๆ เช่น ตัวแปร key, ccNum, dist, visited ซึ่งจะเป็นตัวแปรสำคัญ (ตามที่เรียนในห้อง) และจะถูกใช้ในการประมวลผลทางกราฟ เช่น BFS และ DFS ต่อไป ตัวอย่างการใช้งาน เช่น vertexList[2] = new Vertex(key); แปลว่าให้บรรจุ Vertex (ที่มีค่า key) ลงใน vertexList ตำแหน่ง index เท่ากับ 2 นั่นเอง

ส่วนตัวแปรที่สองนั้นมีชื่อว่า adjacencyList (ซึ่งเป็นตัวแปรชนิด array of LinkedList<Integer> หรือ LinkedList<Integer>[]) โดยตัวแปรนี้จะทำหน้าที่บรรจุข้อมูลว่า Vertex ที่กำลังพิจารณาอยู่นี้ประชิดอยู่กับ Vertex อื่น ๆ ได้อย่าง ยกตัวอย่าง เช่น คำสั่ง LinkedList<Integer> list = adjacencyList[2]; แปลว่า list นี้จะบรรจุตัวเลข Index ของ Vertex อื่น ๆ ที่ประชิดอยู่กับ Vertex(2)

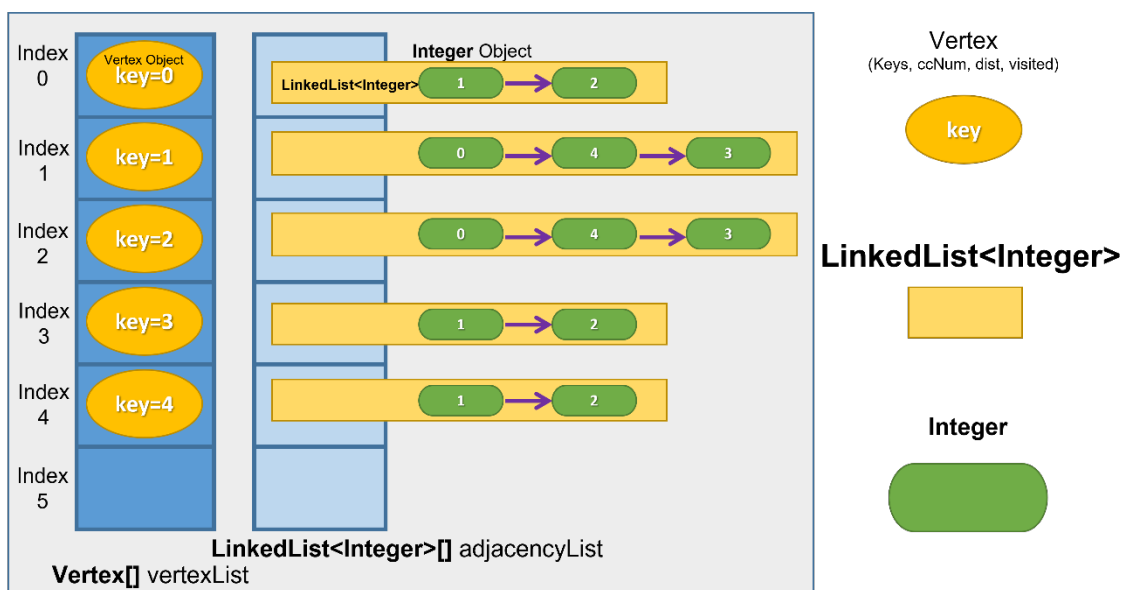
Array ที่ชื่อ vertexList และ adjacencyList ถูกออกแบบมาให้มีขนาดเท่ากัน ทำให้ตำแหน่ง Index เดียวกัน จะอ้างถึงข้อมูล Vertex ตัวเดียวกัน

เพื่อให้เกิดความเข้าใจในโมเดลนี้ อาจารย์ขอยกตัวอย่างแผนภาพกราฟดังต่อไปนี้



ซึ่งเมื่อแปลงเป็นโครงสร้างข้อมูลตามการบ้านนี้แล้ว กราฟดังกล่าวจะมีหน้าตาเป็นแบบนี้

Graph Object



จะเห็นได้ว่า Vertex ที่มี Key=0 เบื้องต้นจะให้อยู่ที่ vertexList[0] หากต้องการจะหาว่า Vertex Key=0 นี้เชื่อมต่อกับ Vertex อะไรบ้างให้ดูใน adjacencyList ที่มี Index ตรงกันคือ adjacencyList[0] ซึ่งจะปรากฏสาย Integer (ขบวนสี่เหลี่ยมที่อยู่ด้านขวา) โดยในแต่ละตัวจะบรรจุข้อมูล Index ของ Vertex ที่เชื่อมต่อกันเอาไว้ (ตามที่ระบุใน vertexList) ดังตัวอย่าง ถ้า Vertex key=0 เชื่อมต่อกับ Vertex key=1 และ Vertex key=2 แล้ว LinkedList ที่ตำแหน่ง adjacencyList[0] ก็จะมีขบวน Node ที่บรรจุ integer เป็น 1 กับ 2 ตามลำดับ อีกตัวอย่างหนึ่ง ก็คือ Vertex key=1 ที่เชื่อมต่อกับ Vertex key=0, Vertex key=4, Vertex key=3 ก็จะมีขบวนของ Integer เป็น 0, 4, 3 ตามลำดับ นศ พอเข้าใจใช่ไหมครับ (ไม่เข้าใจให้ chat ถาม)

นักศึกษาจะสามารถสังเกตได้ว่า ถ้าหาก Vertex key=0 เชื่อมต่อกับ Vertex key=1 ตามที่ระบุไว้ใน adjacencyList[0] แล้ว Vertex key=1 ก็จะเชื่อมต่อกับ Vertex key=0 ตามที่ระบุไว้ใน adjacencyList[1] ด้วยเช่นกัน ความสัมพันธ์นี้จะไปอย่างสมมาตร ดังนั้นเวลานักศึกษา add 1 ไปยัง List[0] แล้ว ก็อย่าลืม add 0 ไปยัง List[1] ด้วยนะครับ (อยู่ในฟังก์ชัน addEdge)

การบ้านนี้นักศึกษาต้องแก้ไข/เพิ่มเติมโค้ดภาษาจาวาของอาจารย์ให้สมบูรณ์เพื่อให้โปรแกรมจะทำงานได้ตามที่กำหนด 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 LinkedList 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 สำหรับการเชื่อมต่อนั้น ให้ add key = v ลงใน LinkedList ของ Vertex u และ เนื่องจากกราฟในวิชานี้เป็นโมเดลที่สมมาตร นักศึกษาอย่างลืมเชื่อม Vertex v กลับไปยัง Vertex u ด้วยนะครับ
 - c. public boolean isConnected(int u, int v) ทำหน้าที่ตรวจสอบว่า Vertex key=u กับ Vertex key=v มีเส้นเชื่อมกันหรือไม่
 - d. public void showAdjacentVertices(int u) ทำหน้าที่แสดงรายการของ Vertex index ทุก ๆ ตัว ที่เชื่อมต่อกับ Vertex key = u [ฟังก์ชันนี้อาจารย์ทำให้เสร็จเรียบร้อยแล้ว ถ้าคุณศึกษาวิธีการเขียนฟังก์ชันนี้ของอาจารย์ดี ๆ บอกเลยว่า คุณจะเข้าใจโครงสร้างข้อมูลของกราฟในการบ้านนี้ทันที]
 - e. public void Explore(Vertex v) ทำหน้าที่สำรวจเชิงลึก แบบ Depth First Search ตามที่เรียนในห้อง โดยเริ่มต้นจาก Vertex v ก่อน วิธีการคือการบิ่สัญลักษณ์ visited = true พร้อมเลข cc เพื่อบ่งชี้ว่าได้มาสำรวจ Vertex นี้แล้ว หลังจากนั้น ก็จะค้นหาต่อว่า Vertex v นี้มีเส้นเชื่อมต่อกับกับ Vertex ใดอีก ถ้ามีก็จะตรวจสอบว่า Vertex นั้นถูก visited หรือยัง ถ้ายังก็เรียก Explore() ซ้ำกับ Vertex นั้นลงไปเรื่อย ๆ แบบ Recursive
 - f. public void DFS() ทำหน้าที่เช็คได้ ทุก ๆ Vertex ที่มีในกราฟว่า Vertex ใดยังไม่ได้ถูกสำรวจบ้าง ถ้าเจอ ก็ให้เรียกฟังก์ชัน Explore() เพื่อประมวลผล ต่อไป ... ทั้งนี้ การที่หลุดออกจากฟังก์ชัน Explore() ออกมาได้ ย่อมแปลความได้ว่า Connected Component นั้น ๆ ถูกประมวลผลครบหมดแล้ว ... ถ้ายังมี

Vertex อื่น ๆ ที่ยังไม่ได้สำรวจ ย่อมแปลว่า ยังมี Connected Components อื่น ๆ อีก ก็ให้เพิ่มค่า cc ไปอีก 1 ... นอกจากนี้ Vertex ใดที่เชื่อมหาต่อกันได้ Vertex นั้นจะถูกจัดกลุ่มให้อยู่ใน Connected Component (CC number) ที่มีเลขเดียวกันอีกด้วย

2. Class HashGraph จะทำหน้าที่คล้ายกับ class Graph ตามที่อธิบายก่อนหน้านี้ นอกจากนี้ อาจารย์ยังสั่งให้ class HashGraph สืบทอดคุณสมบัติ (extends) ของ class Graph มาทั้งหมดอีกด้วย ทั้งนี้ สิ่งที่แตกต่างกันออกไป คือ class HashGraph จะรับ key เข้ามาเป็น String แทนที่จะเป็นจำนวนเต็ม ดังนั้น 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 ถ้า นศ เริ่มต้นการบ้านนี้ด้วยการทำ [หัวข้อที่ 1] นศ ก็จะทำในส่วนนี้เสร็จเรียบร้อยแล้ว นศ เพียงแต่ Copy โค้ดที่ทำไว้แล้วนำมาปรับเข้ากับข้อนี้ ในส่วนของฟังก์ชันที่ชื่อว่า polyHash() กับ getListIndex()

คลาส HashGraph จะมีฟังก์ชันที่สำคัญเพิ่มเติมดังนี้

- a. public static int polyHash(String s, int p, int x) ทำหน้าที่แฮช String s ด้วยอัลกอริทึม Polynomial Hashing ตามที่เรียนในห้อง ... นศ สามารถ Copy โค้ดที่ทำใน [หัวข้อที่ 1] มาลงตรงนี้ได้เลย
- b. public int getListIndex(String s) ทำหน้าที่เรียกฟังก์ชัน polyHash() เพื่อแฮช String อีกที แล้วทำ Cardinality Fix (ปรับตัวเลข เพื่อนำข้อมูลลงตาราง ซึ่งก็คือการทำ mod m นั้น) ถ้าหากว่าตัวเลข Index ที่ได้ ชนเข้ากับข้อมูลที่มีอยู่แล้วก็แก้ไขด้วย Quadratic Probing ตามที่เรียนในห้อง ... นศ สามารถ Copy โค้ดที่ทำใน [หัวข้อที่ 1] แล้วนำมาปรับเข้ากับฟังก์ชันนี้ได้เลย
- c. public void addVertex(String key) ทำหน้าที่คล้าย addVertex() ของ Graph เพียงแต่คุณต้องแฮช String key เพื่อให้ได้ index ที่ว่างก่อนนำข้อมูลไปใส่
- d. public void addEdge(String source, String destination) ทำหน้าที่คล้าย addEdge() ของ Graph เพียงแต่คุณต้องแฮช String source กับ destination ก่อน ตัวเลขที่ได้คุณสามารถเรียกใช้ super.addEdge() ได้เลย หรือคุณจะ Copy โค้ดจาก Graph มาลงก็ได้
- 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

- การสร้าง Queue ให้ใช้คำสั่ง Queue<Vertex> q = new LinkedList();
- การ Enqueue และ Dequeue ให้ใช้คำสั่ง q.add() และ q.remove() ตามลำดับ
- สำหรับการนิยาม Infinity ขอให้นักศึกษาใช้ Integer.MAX_VALUE แทนครับ

- f. public Stack<Vertex> getShortestPathList(Vertex S, Vertex U) ทำหน้าที่สร้างเส้นทางที่สั้นที่สุด เริ่มจาก Vertex U ย้อนกลับไปยัง Vertex S โดยวนลูป push Vertex ลงใน Stack ตั้งแต่ปลายทางกลับมาจนถึงทาง ผ่านตัวแปรที่ชื่อว่า prev ซึ่งตัวแปรดังกล่าวจะต้องถูกกำหนดค่ามาแล้วจากฟังก์ชัน BFS() Psuedocode ของฟังก์ชันนี้อยู่ใน Slide ฝาก นศ แวะไปดูด้วย

- การสร้าง Stack ให้ใช้คำสั่ง `Stack<Vertex> stack = new Stack();`

- การ Push และ Pop ให้ใช้คำสั่ง `stack.push(U)` และ `stack.pop(U)` ตามลำดับ

g. `public void printShortestPath(String s_str, String u_str)` ทำหน้าที่แสดงเส้นทางที่สั้นที่สุดจากเมืองที่ชื่อว่า `s_str` เดินทางไปจนถึงเมืองที่ชื่อว่า `u_str` โดยขั้นตอนการทำงานจะเป็นดังนี้

- ให้ทำการแฮช keys ที่เป็น String (`s_str` กับ `u_str`) ให้เป็น index เสียก่อน ผ่านฟังก์ชัน

`getListIndex()`

- หลังจากนั้น ให้เอา index ที่ได้ไปถึง Vertex ออกมาจาก `vertexList` ก็จะได้ Vertex `S` กับ Vertex `U`

- ให้ทำ BFS ที่ Vertex `S` เพื่อระบุว่า Vertex อื่น ๆ ที่ประชิดกันนี้ มีระยะทางและเส้นทางมาหา Vertex `S` เป็นอย่างไร

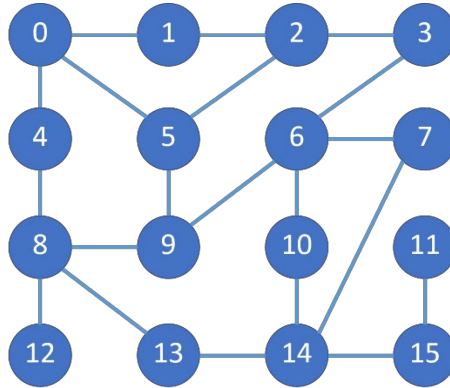
- เรียกใช้ฟังก์ชัน `getShortestPathList(S, U)` เพื่อสร้างเส้นทางจาก `U` ย้อนกลับมา `S` ผลลัพธ์ที่ได้จะออกมาเป็น Stack

- ให้ `nc` pop Stack ที่ว่าออกทีละอัน ผลลัพธ์จากการ pop ก็จะเป็นเส้นทางจากเมือง `S` ไปยังเมือง `U` ที่ถูกต้องครับ

3. สำหรับรายละเอียดของ Class อื่น ๆ หรือฟังก์ชันอื่น ๆ ขอให้นักศึกษา ศึกษาเองตามตัวอย่างในหน้าถัดไป

ตัวอย่างการทำงานที่ 1

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



เมื่อนักศึกษาแก้ไขค้ดตั้งตนอาจารย์ได้เสร็จจ้ลนแล้ว ค้ดตั้งตอไปนี้ ครรที่จะสามารถสร้ง 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);  
}
```

เมื่อเราต้องการใช้คำว่า Vertex key = 0 มี Vertex ที่เป็นเพื่อนบ้านกัน (มี Edge เชื่อมต่อหากัน) มีอะไรบ้าง นักศึกษาสามารถที่จะเรียกคำลั้ง

```
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

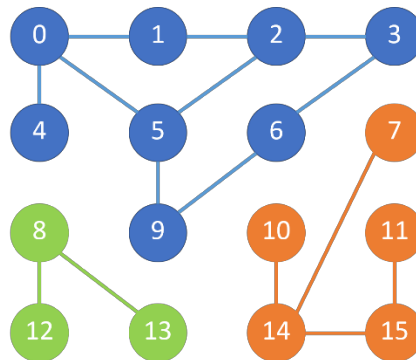
0/1 -> 1/1 -> 2/1 -> 5/1 -> 9/1 -> 6/1 -> 3/1 -> 7/1 -> 14/1 -> 10/1 -> 13/1 -> 8/1 -> 4/1 -> 12/1 -> 15/1 -> 11/1 ->

ผลลัพธ์จะรายงานสองตัวเลขต่อหนึ่ง Vertex คั่นโดย / (slash) โดยตัวเลขแรก แสดงถึงค่า key ของ Vertex ส่วนตัวเลขที่สองคือ Connected Component Number การที่ทุกค่าให้ Connected Component Number เป็นค่าเดียวกันหมดแปลว่า ทุก ๆ Vertex มีเชื่อมไปมาหากันหมด นับเป็นวัฏจักรอันใหญ่ได้หนึ่งก้อน ว่าชั้น

การเดินทางแบบ DFS จาก Vertex แรกไปยัง Vertex สุดท้าย สามารถอ่านได้จากตัวเลขตัวหน้า ดังตัวอย่างคือ 0, 1, 2, 5, 9, 6, 3, 7, 14, 10, 13, 8, 4, 12, 15, 11 การเดินทางแบบนี้ก็จะ เป็น DFS ครับ

ตัวอย่างการทำงานที่ 2

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



โค้ดดังต่อไปนี้คือวิธีที่สามารถสร้าง 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("Number of connected component = " + (graph.cc-1));
```

ผลลัพธ์ที่ได้คือ

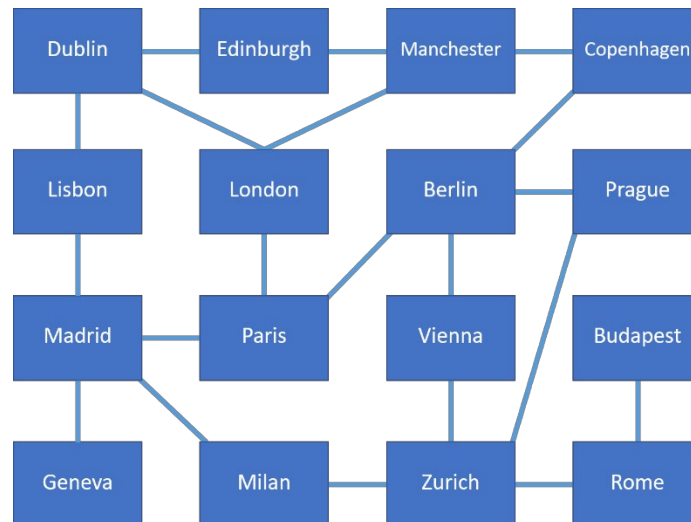
```
0/1 -> 1/1 -> 2/1 -> 5/1 -> 9/1 -> 6/1 -> 3/1 -> 4/1 -> 7/2 -> 14/2 -> 10/2 -> 15/2 -> 11/2 -> 8/3 -> 13/3 -> 12/3 ->  
Number of connected components = 3
```

ผลลัพธ์จะแสดงให้เห็นว่า Vertex แต่ละอันนั้นอยู่ในกลุ่มไหน ซึ่งจากรูป จะมีอยู่สามกลุ่มและผลลัพธ์ก็แสดงให้เห็นว่าโปรแกรมสามารถนับจำนวนกลุ่มได้ถูกต้อง สังเกตด้วยว่าในการเดินทางในกลุ่มเดียวกัน จะเป็นการเดินทางแบบ DFS โดยเริ่มจาก Vertex แรกสุดที่อยู่ในลิสต์

คำถามต่อไปคือ ทำไมในโค้ดอาจารย์ต้องลบหนึ่งออกจาก graph.cc ด้วย เพื่อที่จะนับจำนวน Connected Components ได้ถูกต้องให้นักศึกษาคิดและนำคำตอบไปตอบในข้อสอบ

ตัวอย่างการทำงานที่ 3

กำหนดให้โครงสร้างข้อมูลกราฟ คือแผนภาพดังต่อไปนี้ โดยมี Vertex ทั้งหมด 16 Vertices และมี Edge ทั้งหมด 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 ที่แฮชได้ เกิดการชน (Collision) โปรแกรมก็จะแก้ปัญหการชนด้วย Quadratic Probing ตามที่เรียนในห้อง อาจารย์กำหนดให้คุณใช้ Parameters ดังต่อไปนี้เพื่อสร้าง Polynomial Hashing ตามที่เรียนในห้อง โดยมี $p = 101111$, $x = 101$, $m = 32$

โค้ดดังต่อไปนี้ควรที่จะสามารถสร้าง Graph ได้ตามกำหนด

```
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]);
}
```

หากต้องการทราบว่าเมืองเหล่านี้ถูกบรรจุไว้ใน array vertexList ที่ตำแหน่งใด เราสามารถเรียกใช้ฟังก์ชัน showVertexList() (อันนี้อาจารย์เขียนให้เสร็จแล้วนะครับ) จะได้ว่า

vertexList[0] contains Geneva	vertexList[8] contains Rome	vertexList[16] contains Lisbon	vertexList[24] contains Edinburgh
vertexList[1] contains Milan	vertexList[9] contains Madrid	vertexList[17] null	vertexList[25] null
vertexList[2] null	vertexList[10] null	vertexList[18] null	vertexList[26] contains Budapest
vertexList[3] contains Dublin	vertexList[11] null	vertexList[19] null	vertexList[27] null
vertexList[4] contains Manchester	vertexList[12] null	vertexList[20] null	vertexList[28] contains Paris
vertexList[5] contains Prague	vertexList[13] contains Copenhagen	vertexList[21] contains Berlin	vertexList[29] null
vertexList[6] contains London	vertexList[14] null	vertexList[22] null	vertexList[30] contains Vienna
vertexList[7] contains Zurich	vertexList[15] null	vertexList[23] null	vertexList[31] null

ตรงนี้ นศ. สามารถนำไปเปรียบเทียบกับผลลัพธ์ของตัวเองได้ ว่าทำ Hashing ถูกต้องหรือไม่ เมื่อถูกต้องแล้วก็ไปต่อกันครับ
หากต้องการเชื่อมเส้นทางของสองเมืองเข้าหากัน ก็เขียนโค้ดเพิ่มเติมว่า

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");

เมื่อเราต้องการเช็คว่ามีเมือง Paris, Zurich, Geneva ติดต่อกับเมืองใดบ้าง นักศึกษาสามารถที่จะเรียกคำสั่งดังต่อไปนี้

graph.showAdjacentVertices(graph.getListIndex("Paris"));
graph.showAdjacentVertices(graph.getListIndex("Zurich"));
graph.showAdjacentVertices(graph.getListIndex("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.printShortestPath("London", "Budapest");  
graph.printShortestPath("Berlin", "Dublin");
```

ผลลัพธ์ที่ได้คือ

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

ซึ่งเมื่อตรวจสอบกับแผนภาพด้านบนก็จะพบว่า เส้นทางที่แสดงนี้เป็นเส้นทางที่สั้นที่สุดแล้ว

หากการบ้านนี้นักศึกษาไม่รู้จะไปยังไง ไม่เข้าใจเลย หรือเสียเวลาการทำการบ้านมากเกินไป ขอให้นักศึกษาเข้ามาปรึกษากับอาจารย์เป็นการด่วน เพื่อที่จะไม่เสียเวลาในการอ่านเตรียมสอบวิชาอื่นต่อไป