งานที่ 4 ข้อ 2

1. **กำหนดคลาส Graph:**
    - กำหนดลิสต์ของ nodes และแผนที่เก็บ edges โดยใช้ HashMap.
    - มีเมธอด **addNode** เพื่อเพิ่ม node ลงในกราฟ.
    - มีเมธอด **addEdge** เพื่อเพิ่มเส้นเชื่อม (edge) ระหว่าง nodes พร้อมกับน้ำหนัก (weight).
2. **กำหนดคลาส Edge:**
    - คลาสนี้เก็บข้อมูลเส้นเชื่อมระหว่าง nodes พร้อมกับน้ำหนัก.
3. **กำหนดคลาส Dijkstra:**
    - กำหนดคลาส Dijkstra ที่ใช้วิธี Dijkstra algorithm.
    - มีตัวแปรเก็บกราฟ, ตารางระยะทาง, ตาราง node ก่อนหน้า, และ PriorityQueue สำหรับการจัดเรียง node ตามระยะทาง.
4. **สร้างกราฟและรัน Dijkstra Algorithm:**
    - ใน **main** method, รับข้อมูลจำนวน nodes และ edges จากผู้ใช้ผ่านทาง Console.
    - เพิ่ม nodes ลงในกราฟ.
    - เพิ่ม edges และน้ำหนัก.
    - รับจุดเริ่มต้นจากผู้ใช้.
    - สร้าง Dijkstra object และเรียก **shortestPath** method เพื่อคำนวณระยะทางที่สั้นที่สุด.
5. **Dijkstra Algorithm:**
    - ใน **shortestPath** method, เตรียมตัวแปรเริ่มต้นและเพิ่ม node ทั้งหมดลงใน PriorityQueue.
    - กำหนดระยะทางของ node เริ่มต้นเป็น 0.
    - ทำลูปจนกว่า PriorityQueue จะไม่ว่าง:
        - ดึง node ที่มีระยะทางน้อยที่สุดออกมา.
        - สำรวจเส้นเชื่อมที่เชื่อมกับ node ปัจจุบัน และอัปเดตระยะทางถ้ามีค่าที่น้อยลง.

- คืนรายการระยะทางทั้งหมด.

6. **Build Path:**
    - ใน **buildPath** method, สร้างเส้นทางจาก node เป้าหมายถอยหลังไปยัง node เริ่มต้น.
    - คืนเส้นทางที่สร้าง.
7. **แสดงผลลัพธ์:**
    - แสดงผลระยะทางทั้งหมดจากจุดเริ่มต้นไปยังทุกๆ node ในกราฟ.

User จะกรอกข้อมูลที่ต้องการ เพื่อสร้างกราฟและหาเส้นทางที่สั้นที่สุดจากจุดเริ่มต้นไปยังจุดปลายทางที่ระบุ. โปรแกรมจะแสดงผลลัพธ์ของระยะทางทั้งหมดจากจุดเริ่มต้นไปยังทุกๆ node ในกราฟ.

psurdo coded

```
กระบวนการ Dijkstra(กราฟ, จุดเริ่มต้น):
    สร้างเซ็ต S ที่ว่างเปล่า
    สร้างอาร์เรย์ระยะทาง dist[] ที่มีค่าเป็นอนันต์
    สร้างอาร์เรย์ parent[] ที่มีค่าเป็น null
    สร้างคิวลำดับความสำคัญ Q

    dist[จุดเริ่มต้น] = 0
    เอ็นคิวจุดเริ่มต้นเข้าไปใน Q

    ขณะที่ Q ไม่ว่าง:
        ปัจจุบัน = ดีคิวจาก Q
        เพิ่ม ปัจจุบัน เข้าไปใน S

        สำหรับทุกรายการเพื่อเพื่อนบ้านในปัจจุบัน:
            หากเพื่อนบ้านไม่ได้อยู่ใน S:
                คำนวณ new_distance = dist[ปัจจุบัน] + น้ำหนัก(ปัจจุบัน, เพื่อนบ้าน)
                หาก new_distance < dist[เพื่อนบ้าน]:
                    อัปเดต dist[เพื่อนบ้าน] เป็น new_distance
                    อัปเดต parent[เพื่อนบ้าน] เป็น ปัจจุบัน
                    เอ็นคิวเพื่อนบ้านเข้าไปใน Q

กระบวนการ BuildPath(parent, เป้าหมาย):
    สร้างรายการทางที่ว่างเปล่า path
```

ขณะที่ เป้าหมาย ไม่เท่ากับ null:

เพิ่ม เป้าหมาย ไปที่ต้น path

เป้าหมาย = parent[เป้าหมาย]

ส่งคืน path

กระบวนการ Main():

numVertices = อ่านจำนวนเต็มจากข้อมูลนำเข้า

กราฟ = สร้างกราฟใหม่

สำหรับ i ตั้งแต่ 1 ถึง numVertices:

vertex = อ่านสตริงจากข้อมูลนำเข้า

กราฟ.addNode(vertex)

numEdges = อ่านจำนวนเต็มจากข้อมูลนำเข้า

สำหรับ i ตั้งแต่ 1 ถึง numEdges:

vertex1, vertex2, weight = อ่านข้อมูลนำเข้า

กราฟ.addEdge(vertex1, vertex2, weight)

จุดเริ่มต้น = อ่านสตริงจากข้อมูลนำเข้า

dijkstra = สร้าง Dijkstra ใหม่ ด้วย กราฟ

result = dijkstra.shortestPath(จุดเริ่มต้น, "D")

```
    พิมพ์ "จุด\tระยะทางจากต้นทาง"
    สำหรับทุกจุดใน กราฟ.getNodes():
        พิมพ์ จุด, "\t", result[จุด]

Main()
```