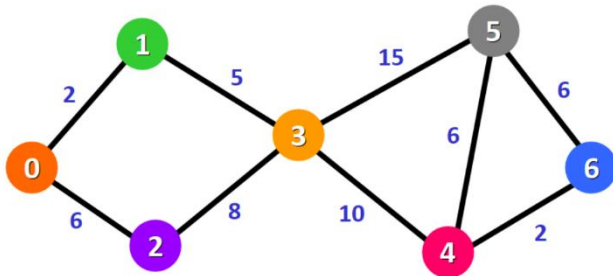


ใบงานที่ 5

วัตถุประสงค์ เพื่อสร้างประสบการณ์การใช้ graph ด้วย adjacency matrix



initial

Distance:

0: 0
1: ∞
2: ∞
3: ∞
4: ∞
5: ∞
6: ∞

city = 0

Distance:

0: 0
1: ~~∞~~ 2
2: ~~∞~~ 6
3: ∞
4: ∞
5: ∞
6: ∞

โจทย์ single source shortest path หมายถึง การหาค่าจาก ต้นทาง ไปแต่ ละเมืองในแผนที่ (graph) ระยะทางที่สั้นที่สุดเป็นเท่าไร

หนึ่งในวิธีการ represent graph คือสร้าง ตาราง 2 มิติ (เรียก adjacency matrix เพราะ adjacency คืออยู่ติดกัน) โดยระบุระยะทางตาม graph ใส่ค่า infinity สำหรับเมืองที่ไม่ได้ (ส่วนของโปรแกรมตัวอย่างใส่ค่า 711 ก็เพียงพอ ต่อการทดสอบ (เพราะมันแพงกว่าค่าในโจทย์) เช่น $\text{adja}[1][3] = \text{adja}[3][1] = 5$ ดูภาพกราฟโจทย์

หลักการคิดคือเราจะพิจารณาตามจำนวนเมือง เช่น รอบแรกพิจารณาจากเมือง 0 (source) โดยเราจะดูทุกเมือง (dest) ว่ามีทางไปได้ไหม $\text{adja}[0][\text{dest}] > 0$ (และไม่ใช่ infinity) ก็คือมี edge จาก source ดังนั้นรอบแรกจะมีเพียง 2 เมือง ที่ได้บันทึกในตาราง (เพราะ dest อื่นเป็น infinity)

รอบถัดไป คือ เรานำมาพิจารณาว่า $\text{dist}[\text{city}] + \text{adja}[\text{city}][\text{dest}]$ มี ระยะทางสั้นกว่า (จากเท่าที่รอบก่อนๆรู้มา) กล่าวคือหากการเดินทางผ่าน city นี้ ทำให้ทาง $\text{dist}[\text{dest}]$ ลดลง เราจึง update $\text{dist}[\text{dest}]$ นี้ภาพว่า เรามี $\text{dist}[\text{สมุทรสงคราม}]$ $\text{dist}[\text{เพชรบุรี}]$ เมื่อ city เป็น นครปฐม เราก็นำมาคำนวณ ว่า $\text{dist}[\text{นครปฐม}] + \text{adja}[\text{นครปฐม}][\text{สมุทรสงคราม}]$ สั้นกว่า $\text{dist}[\text{สมุทรสงคราม}]$ ที่เคยคำนวณไว้หรือไม่

ประเด็นการพิสูจน์ ว่าสุดท้ายแล้วจะได้สั้นสุดทุกเมืองได้จริงหรือเปล่า นั้นเป็น เพราะ ลูปนอก เรานำทุกเมืองมาพิจารณา และลูปใน เรานำเมืองนั้นๆ(dest) มา เปรียบกับทุกปลายทาง (ลูปนอกที่หา shortest distance ไปแล้ว)

แอปพลิเคชันแนวนี้คือ หาเส้นทางที่เหมาะสมที่สุดที่ส่งข้อมูลข้ามอินเทอร์เน็ต

Distance:

0: 0
1: ~~∞~~ 2
2: ~~∞~~ 6
3: ~~∞~~ 7 from (5 + 2) vs. 14 from (6 + 8)
4: ∞
5: ∞
6: ∞

city = 1, 2

Distance:

0: 0
1: ~~∞~~ 2
2: ~~∞~~ 6
3: ~~∞~~ 7
4: ~~∞~~ 17 from (2 + 5 + 10)
5: ~~∞~~ 22 from (2 + 5 + 15)
6: ∞

city = 3

Distance:

0: 0
1: ~~∞~~ 2
2: ~~∞~~ 6
3: ~~∞~~ 7
4: ~~∞~~ 17
5: ~~∞~~ 22 vs. 23 (2 + 5 + 10 + 6)
6: ~~∞~~ 19 from (2 + 5 + 10 + 2)

city = 4

Distance:

0: 0
1: ~~∞~~ 2
2: ~~∞~~ 6
3: ~~∞~~ 7
4: ~~∞~~ 17
5: ~~∞~~ 22
6: ~~∞~~ 19

city = 5, 6

ภาพประกอบจาก <https://www.freecodecamp.org/news/dijkstras-shortest-path-algorithm-visual-introduction/>

ลำดับการประมวลผลใน page ไม่ตรงกับส่วนของโปรแกรมด้านล่าง เพราะทำการ simplify code

```

[[[]] adja ={{ 0, 2, 6, 711, 711, 711, 711 },
              { 2, 0, 711, 5, 711, 711, 711 },
              { 6, 711, 0, 8, 711, 711, 711 },
              { 711, 5, 8, 0, 10, 15, 711 },
              { 711, 711, 711, 10, 0, 6, 2 },
              { 711, 711, 711, 15, 6, 0, 6 },
              { 711, 711, 711, 711, 2, 6, 0 } };

[] dist ={ 0, 711, 711, 711, 711, 711, 711 };

for (int city = 0; city < adja.length; city++) {
    for (int dest = 1; dest < adja.length; dest++) {
        if (dist[dest] < 711)
            println("dest " + dest + " dist " + dist[dest]); //q1
        if ( /* q2 */ ) {
            tmp++; // System.out.println(" new cost for " + dest);
            dist[dest] = dist[city] + adja[city][dest];
        }
    }
    System.out.println("-----");
}
System.out.println(tmp); /* q3 */
System.out.println(Arrays.toString(dist)); /* q4 */

```

คำสั่ง

q1 แสดงผลการทำงาน

q2 เติมส่วนของโปรแกรม

q3 & q4 แสดงผลการทำงาน

กำหนดส่ง TBA