มีการเปลี่ยนจาก Queue เป็น Stack ที่

บรรทัดที่ 36 จาก Queue q = new ArrayQueue() → Stack s = new ArrayStack()

บรรทัดที่ 37 จาก q.enqueue(source) → s.push(source)

บรรทัดที่ 39 จาก q.dequeue() → s.pop()

บรรทัดที่ 50 จาก findPath(Queue q, ...) → findPath(Stack s, ...)

บรรทัดที่ 55 จาก q.enqueue(...) → s.push(...)

จากบรรทัดที่ 36 จากเดิมจะเป็นการเรียก class ArrayQueue มาใช้ แต่เปลี่ยนเป็น ArrayStack เพื่อให้ใช้ method จาก class ArrayStack ได้ ซึ่งส่งผลต่อการเรียกใช้ method ในบรรทัดที่ 37 ซึ่งเป็นการเอาค่าใส่ไปในด้านหลังของ Queue และบนสุดออกจาก Stack ในบรรทัดต่อมาเป็นการเช็คเงื่อนไขใน while loop ถ้า Queue หรือ Stack นั้นมีค่าถูกเก็บอยู่ถึงจะวนทำงานใน loop โดยการทำงานใน loop จะเริ่มจากการนำค่าที่อยู่แรกสุดของ Queue/ตัวที่อยู่บนสุดของ Stack ออกมา ซึ่งก็คือ source ที่เราใส่เข้าไปในบรรทัดที่ 37 ซึ่งจากโจทย์มีค่า row = 0, col = 0 จากนั้นเช็คเงื่อนไขว่ายังไม่ถึงค่าเป้าหมาย(target) ถ้ายังไม่ถึงให้ทำงานต่อโดยการเรียก method expand ซึ่งเป็นการตรวจสอบค่าทั้ง 4 ทิศเพื่อเดินต่อ โดยถ้าค่าไม่ใช่ block ขอบของกระดานก็จะเดินต่อไปได้จากนั้นก็จะนำค่า row และ col ของตำแหน่งที่ expand มาใส่เข้าไปใน queue/stack เพื่อนำทางไปถึงเป้าหมาย

ตัวอย่างตาราง

Row \ Col	0	1	2	3
0	(start)	-1	-1	-1
1	-1	-9	-1	(target)
2	-1	-1	-9	-1
3	-1	-1	-1	-9

	Queue	Stack
Start	-	-
Add	(0,0)	(0,0)
Out	-	-
Add	(1,0)	(1,0)
Add	(1,0), (0,1)	(1,0), (0,1)
Out	(0,1)	(1,0)
Add	(0,1), (2,0)	(1,0), (0,2)
Out	(2,0)	(1,0)
Add	(2,0), (0,2)	(1,0), (0,3)
Add	(2,0), (0,2)	(1,0), (0,3), (1,2)

จากตารางเปรียบเทียบการทำงานของ Queue และ Stack จะเห็นว่าทั้งคู่มีการทำงานที่แตกต่างกันซึ่งจะให้ผลลัพธ์ที่ไม่เหมือนกัน เนื่องจาก Queue มีการนำค่าที่เก็บไว้แรกสุดมาใช้ก่อนแต่ Stack มีการใช้ค่าที่เก็บในลำดับสุดท้าย ซึ่งในการใช้ Queue สำหรับโจทย์ข้อนี้จะมีการดำเนินงานที่ครอบคลุมมากกว่า Stack เพราะ Queue จะเช็คทุกทิศทางรอบตัวเองแต่ Stack ในข้อนี้จะมุ่งไปทางใดทางหนึ่ง