



Project Report

Data Structure and Algorithms

เรื่อง การแก้ปัญหา Lights Out Puzzle

เสนอ

รศ.ดร.รังสีพรรณ มฤคทัต

(Assoc.Prof. Rangsipan Marukatat, Ph.D)

จัดทำโดย

นาย	ณัฐวิทย์	เกิงฝาก	รหัสนักศึกษา	6213125
นาย	วสวัตต์	เพ็งประโคน	รหัสนักศึกษา	6213132
นางสาว	ณิชารีย์	เฉลิมสุขศรี	รหัสนักศึกษา	6213198
นาย	ภูวิช	โรจนกนกโชค	รหัสนักศึกษา	6213209

Department of computer Engineering

Faculty of Engineering, Mahidol University

คำนำ

รายงานฉบับนี้เป็นส่วนหนึ่งของวิชา Data Structure and Algorithms (EGCO 221) โดยคณะผู้จัดทำได้จัดทำขึ้นเพื่ออธิบายการทำงานของโปรแกรมแก้ปัญหา Light out puzzle ซึ่งเป็นโปรแกรมที่ใช้ในการคำนวณหาเส้นทางที่สั้นที่สุดที่ใช้ในการปิดไฟทุกดวงที่ถูกเปิดขึ้น โดยใช้ Data Structure และ Algorithms มาประยุกต์ใช้ในการแก้ปัญหานี้ โดยในรายงานประกอบไปด้วยคู่มือการใช้งานโปรแกรม การอธิบายการทำงานของ Code และ Algorithms รวมไปถึงข้อจำกัดต่าง ๆ ของโปรแกรม

คณะผู้จัดทำขอขอบพระคุณ รศ.ดร.รังสิพรรณ มฤคทัต ผู้ให้ความรู้ และแนวทางในการศึกษา สุดท้ายนี้ทางคณะผู้จัดทำหวังว่ารายงานฉบับนี้ จะให้ความรู้และประโยชน์ไม่มากนักแก่ผู้อ่านทุกท่าน หากมีข้อผิดพลาดประการใด ผู้จัดทำจะขอน้อมรับไว้ และขออภัยมา ณ ที่นี้ด้วย

คณะผู้จัดทำ

สารบัญ





หัวข้อ	หน้า
คำนำ	ก
สารบัญ	๗
เกี่ยวกับโปรแกรม	1
คู่มือการใช้งาน (User Manual)	2
โครงสร้างข้อมูล (Data Structure)	6
- Package java.util.*	6
- Package org.jgrapht	6
- Class PuzzleState	7
- Class lightSolver	8
- Class light (Main Class)	9
อัลกอริทึมที่ใช้ในการแก้ปัญหา	10
แผนผังการทำงานของโปรแกรม (Flow Chart)	12
- แผนผังการทำงานของ Main Function	12
- แผนผังการทำงานของ Solve Function	13
อธิบายการทำงานของโปรแกรม	14
- ส่วน Main class (Class light)	14
- ส่วนการประมวลผลการแก้ปัญหา (Class lightSolver ใน Solve Function)	17
ตัวอย่างแสดงการประมวลผล	18
Runtime ของโปรแกรม	30
ข้อจำกัดของโปรแกรม	32
แหล่งอ้างอิง	36

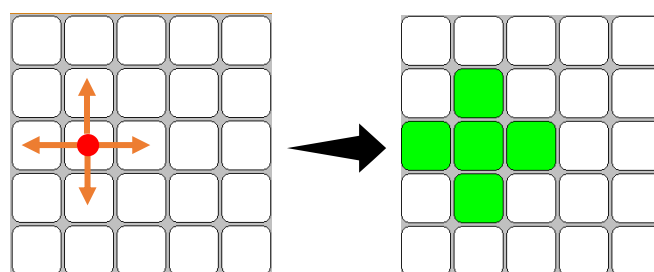
เกี่ยวกับโปรแกรมนี้

โปรแกรมนี้เป็นโปรแกรมที่ใช้ในแก้ปัญหา Light out puzzle ซึ่งเป็นโปรแกรมที่ใช้ในการคำนวณหาเส้นทางที่สั้นที่สุดที่ใช้ในการปิดไฟทุกดวงที่ถูกเปิดขึ้น โดยในการกดเปิด/ปิดไฟแต่ละดวงนั้น จะทำการเปิด/ปิดไฟดวงที่อยู่ข้าง ๆ ด้วยเสมอ โดยมีกฎและวิธีการเล่นตัวอย่างต่อไปนี้

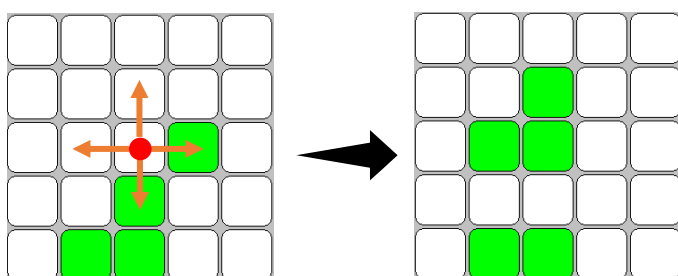
1. หากทำการกดปุ่มที่ไม่ได้อยู่ชิดขอบ จะทำการเปิด/ปิดไฟดวงนั้น และไฟที่อยู่ข้าง ๆ ทั้ง 4 ดวงให้เปลี่ยนเป็นไฟในสถานะตรงข้ามจากเดิม ตามลูกศรสีส้ม ดังตัวอย่างที่ 1 และตัวอย่างที่ 2
2. หากทำการกดปุ่มที่อยู่ชิดขอบ จะทำการเปิด/ปิดไฟดวงนั้น และไฟที่อยู่ข้าง ๆ เหมือนกับข้อ 1 แต่ขอบเขตที่กำหนดอยู่ในกรอบเท่านั้น จะไม่วกกลับไปเปิดที่ขอบอีกฝั่งดังตัวอย่างที่ 3 และตัวอย่างที่ 5
3. การแก้ปัญหานี้จะสำเร็จต่อเมื่อเรากดเปิด/ปิดไฟ จนได้ผลลัพธ์เป็นไฟดับทุกดวง ดังตัวอย่างที่ 4

สัญลักษณ์ที่ใช้ในการอธิบาย

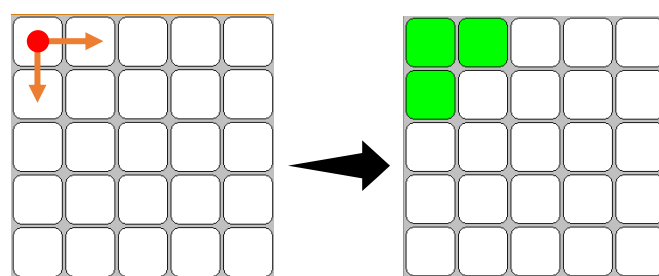
-  : แทนไฟเปิด
-  : แทนไฟปิด
-  : แทนจุดที่ทำการกดปุ่ม
-  : แทนเส้นทางที่ไฟเกิดการเปลี่ยนแปลงเมื่อกดปุ่ม



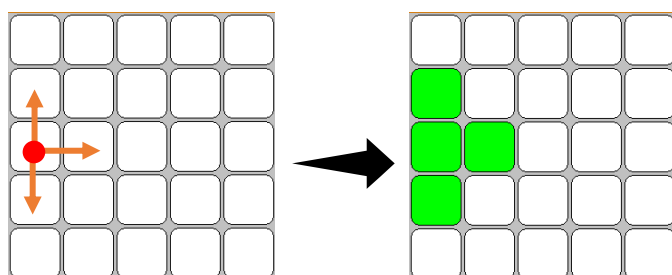
ตัวอย่างที่ 1



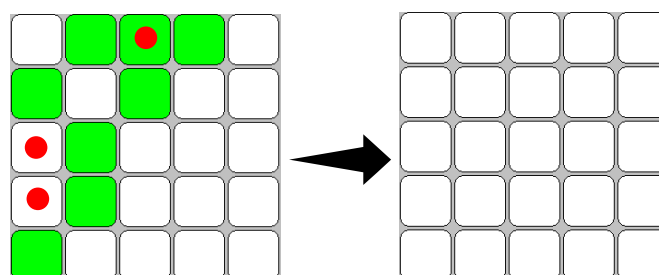
ตัวอย่างที่ 2



ตัวอย่างที่ 3



ตัวอย่างที่ 4



ตัวอย่างที่ 5

คู่มือการใช้งานโปรแกรม (User Manual)

- เมื่อโปรแกรมเริ่มทำงาน จะให้ User ป้อนค่าขนาดของตาราง Puzzle ที่ต้องการประมวลผล ซึ่งขนาดของตารางที่รับค่าเข้ามามีค่าเท่ากับ $N \times N$ โดยโปรแกรมกำหนดไว้ว่าขนาดของตารางที่รับค่าเข้ามาจะต้องมีค่าน้อยกว่า 3x3 และค่าที่ป้อนเข้ามาจะต้องเป็นตัวเลขที่อยู่ในช่วง (-2147483647, 2147483647)

```

=====Input puzzle data=====
Input puzzle size(N) :
1                                     ✗ Invalid input because number less than 3.
***** !!! WARNING !!! *****
          Size of puzzle must be more than 3, please try again.
*****

Input puzzle size(N) :
-12                                  ✗ Invalid input because number less than 3.
***** !!! WARNING !!! *****
          Size of puzzle must be more than 3, please try again.
*****

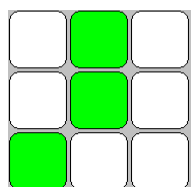
Input puzzle size(N) :
five                                 ✗ Invalid input because input isn't integer.
***** !!! WARNING !!! *****
          Type of input error (Integer only) or
          A number is out of range(3,2147483647)
          Please try again.
*****

Input puzzle size(N) :
100000000000000000000000000000000 ✗ Invalid input because number is out of range
***** !!! WARNING !!! *****
          Type of input error (Integer only) or
          A number is out of range(3,2147483647)
          Please try again.
*****

Input puzzle size(N) :
3                                     ✓ Valid input

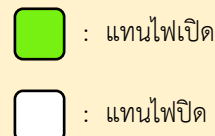
```

2. จากนั้นให้ User ทำการป้อนสถานะเริ่มต้นของไฟแต่ละดวงว่าไฟกำลังเปิด/ปิดอยู่ โดยกำหนดให้ป้อนค่าเป็นตัวเลข Bit string ตามลำดับเริ่มจากไฟตำแหน่งมุมซ้ายแฉวบนสุด แล้วไล่จากซ้ายไปขวา และจากแฉวบนลงแฉวล่าง ซึ่งความยาวของตัวเลข Bit String นั้นมีค่าเท่ากับจำนวนขนาดของตารางที่รับค่าเข้ามา ($N \times N$) โดยหากไฟตำแหน่งนั้นดับให้มีค่าเป็น 0 และหากไฟดวงนั้นเปิดให้ป้อนมีค่าเป็น 1 เช่น



Bit string = 010010100

สัญลักษณ์ที่ใช้ในการอธิบาย



ความยาวของตัวเลข Bit String = $N \times N = 3 \times 3 = 9$ bits

Input puzzle size(N):

3

Input puzzle states (9 bits, left to right, line by line):

101011

✗ Invalid input because bit string isn't 9 bits

***** !!! WARNING !!! *****

Invalid state input!, please input 9 bits

Input puzzle states (9 bits, left to right, line by line):

101110100101

✗ Invalid input because bit string isn't 9 bits

***** !!! WARNING !!! *****

Invalid state input!, please input 9 bits

Input puzzle states (9 bits, left to right, line by line):

210013311

✗ Invalid input because a number of input must be 0 or 1

Invalid state input!, please input only 0 or 1

***** !!! WARNING !!! *****

Type of input error (Integer only) or

A number of input must be only 0 or 1

Please try again.

Input puzzle states (9 bits, left to right, line by line):

onezero01010one

✗ Invalid input because a number of input must be 0 or 1

Invalid state input!, please input only 0 or 1

***** !!! WARNING !!! *****

Type of input error (Integer only) or

A number of input must be only 0 or 1

Please try again.

Input puzzle states (9 bits, left to right, line by line):

110001110

✓ Valid input

3. หลังจากที่ User กรอกขนาดของ Puzzle และสถานะของไฟแต่ละดวงเรียบร้อยแล้ว โปรแกรมจะทำการประมวลผลตำแหน่งของไฟที่ต้องกด เพื่อที่จะทำให้ไฟทุกดวงดับ และถาม User ว่าต้องการใช้งานโปรแกรมอีกรอบหรือไม่ ถ้าหากต้องการใช้งานโปรแกรมอีกรอบให้ป้อน Y/y แต่ถ้าต้องการจบการทำงานของโปรแกรมให้ป้อน N/n ถ้า User ป้อนอย่างอื่นนอกเหนือจากนี้ โปรแกรมจะให้ User ป้อนใหม่อีกครั้ง

```

=====Input puzzle data=====
Input puzzle size(N) :
3
Input puzzle states (9 bits, left to right, line by line):
110001110
=====Puzzle button layout=====

```

1	2	3
4	5	6
7	8	9

แสดงตำแหน่งหมายเลขของไฟแต่ละดวง

```

=====Puzzle board=====
Puzzle problem:
Bit string = 110001110

```

1	1	0
0	0	1
1	1	0

แสดงสถานะเริ่มต้นของไฟแต่ละดวงที่รับค่ามาจาก user

```

=====Puzzle solution=====
Puzzle solution: 4->5
2 moves to turn off all lights
Show solution:
>>> Move 1 : turn on button 4
Bit string = 010111010

```

0	1	0
1	1	1
0	1	0

แสดงเส้นทางที่สั้นที่สุดที่ใช้ในการปิดไฟทุกดวงที่ถูกเปิดขึ้น โดยจะแสดงผลลำดับการเปิด/ปิดไฟ ด้วยตำแหน่งหมายเลขของไฟแต่ละดวง และแสดงจำนวนการ move ทั้งหมดที่ใช้ในการเปิด/ปิดไฟ

```

>>> Move 2 : turn off button 5
Bit string = 000000000

```

0	0	0
0	0	0
0	0	0

แสดงตำแหน่งหมายเลขของไฟที่ถูกเปิด/ปิด และการเปลี่ยนแปลงของไฟแต่ละดวงในแต่ละ State ของ move ทั้งหมดตามลำดับ

```

=====The puzzle has been solved!=====
Do you want to repeat ? (Y/N)
Yes ❌ Invalid input because input must be Y/y or N/n
Invalid input!, Y to repeat or N to quit program
Do you want to repeat ? (Y/N)
yes ❌ Invalid input because input must be Y/y or N/n
Invalid input!, Y to repeat or N to quit program
Do you want to repeat ? (Y/N)
y ✅ Valid input
=====Input puzzle data=====
Input puzzle size(N):
|

```

หากต้องการใช้งานโปรแกรมอีกครั้งให้ป้อน Y/y แต่ถ้าต้องการจบการทำงานของโปรแกรมให้ป้อน N/n

4. ถ้าไม่มีวิธีที่จะดับไฟทุกดวงได้ โปรแกรมจะแสดงผลว่าไม่มีคำตอบ แล้วจึงถาม User ว่าต้องการจะใช้งานโปรแกรมอีกรอบหรือไม่

```

=====Puzzle board=====
Puzzle problem:
Bit string = 1101011101011101
-----
| 1 | 1 | 0 | 1 |
-----
| 0 | 1 | 1 | 1 |
-----
| 0 | 1 | 0 | 1 |
-----
| 1 | 1 | 0 | 1 |
-----
=====Puzzle solution=====
Puzzle solution: No solution
=====The puzzle has been solved!=====
Do you want to repeat ? (Y/N)

```

โปรแกรมจะแสดงผลว่าไม่มีคำตอบ

5. ถ้าขนาดของ Puzzle ใหญ่เกินไปจนทำให้เกิด OutOfMemoryError โปรแกรมจะแสดงผลว่าไม่สามารถหาคำตอบแล้ว แล้วจึงถาม User ว่าต้องการจะใช้งานโปรแกรมอีกรอบหรือไม่

```

=====Input puzzle data=====
Input puzzle size(N):
5
Input puzzle states (25 bits, left to right,line by line):
1010000010010000001000000
=====Puzzle button layout=====
-----
| 1 | 2 | 3 | 4 | 5 |
-----
| 6 | 7 | 8 | 9 | 10 |
-----
| 11 | 12 | 13 | 14 | 15 |
-----
| 16 | 17 | 18 | 19 | 20 |
-----
| 21 | 22 | 23 | 24 | 25 |
-----
=====Puzzle board=====
Puzzle problem:
Bit string = 1010000010010000001000000
-----
| 1 | 0 | 1 | 0 | 0 |
-----
| 0 | 0 | 0 | 1 | 0 |
-----
| 0 | 1 | 0 | 0 | 0 |
-----
| 0 | 0 | 0 | 1 | 0 |
-----
| 0 | 0 | 0 | 0 | 0 |
-----
***** !!! WARNING !!! *****
                Error! Out of memory
                Can't find solution of the puzzle
*****
Do you want to repeat ? (Y/N)

```

ขนาดของ Puzzle ใหญ่เกินไปจนทำให้เกิด
OutOfMemoryError

โครงสร้างข้อมูล (Data Structures)

1. Package java.util.*

```
import java.util.*;
```

เป็นการเรียกใช้งานไลบรารีของภาษา Java ที่อยู่ใน Package java.util และใน Collection ซึ่งโดยส่วนมากนั้นสืบทอดมาจาก Abstract class และมีการ Implement บาง Interface มา รวมกันไว้ โดย Collection นั้นเป็นการเก็บข้อมูลแบบไดนามิกส์ นั้นหมายความว่าเราสามารถเพิ่มข้อมูลเข้าไปใน Collection ได้ไม่จำกัด โดยที่ไม่ต้องกำหนดขนาดสูงสุดในการเก็บข้อมูลล่วงหน้า Collection มีเมธอดที่ทำงานด้วยอัลกอริทึมที่มีประสิทธิภาพเป็นจำนวนมากให้สามารถใช้งานได้ทันที โดยที่ไม่ต้องเขียนขึ้นอีก Collection ในภาษา Java เบื้องต้น เช่น ArrayList, Stack และ Deque ซึ่งอัลกอริทึมที่เราเลือกใช้ในโปรแกรมคือ ArrayList และ ArrayDeque (ซึ่งมีพื้นฐานมาจาก Array และ Deque)

2. Package org.jgrapht

```
import org.jgrapht.Graph.*;
import org.jgrapht.Graph;
import org.jgrapht.Graphs;
import org.jgrapht.alg.interfaces.ShortestPathAlgorithm;
import org.jgrapht.alg.shortestpath.DijkstraShortestPath;
import org.jgrapht.graph.DefaultEdge;
import org.jgrapht.graph.SimpleDirectedGraph;
```

เป็นการเรียกใช้งานไลบรารีของภาษา Java ที่อยู่ใน Package org.jgrapht ซึ่งนำมาใช้ในการสร้างกราฟ และการดำเนินการต่าง ๆ บนกราฟ ซึ่งประเภทของกราฟที่เราเลือกใช้ในโปรแกรมนี้นี้คือ Simple directed graph โดยอัลกอริทึมที่เราเลือกใช้ ได้แก่

1. **Breadth First Search Algorithm** : ใช้ในการไล่ตรวจสอบ Vertex ต่าง ๆ ภายในกราฟ
2. **Dijkstra's Algorithm** : ใช้ในการหาเส้นทางที่สั้นที่สุดที่ใช้ในการปิดไฟทุกดวงที่ถูกเปิดขึ้นที่เลือกใช้อัลกอริทึมนี้

3. Class PuzzleState

```
public class PuzzleState{
    public String state;
    public int button;
    public int code;

    public PuzzleState(String s) {...4 lines }

    public String getState() {...3 lines }

    public int getButton() {...3 lines }

    public void setButton(int n) {...3 lines }

    public boolean equals(Object o) {...4 lines }

    public int hashCode() {...3 lines }
}
```

Class PuzzleState เป็น Class ที่ทำหน้าที่ในการเก็บตาราง Puzzle ที่จะทำการประมวลผล โดยจะมีตัวแปร ดังนี้

1. String state : เก็บสถานะของไฟทั้งหมดในตารางในรูปแบบของ Bit string
2. int button : เก็บค่าว่า state นี้เกิดมาจากการกดปุ่มใดบนตาราง Puzzle
3. int code : แปลงตัวแปร state ให้เป็นเลขฐาน 10 เพื่อนำไปใช้เปรียบเทียบใน Function equal และ hashCode

และมี method ที่สำคัญ ดังนี้

1. getState() : คืนค่าสถานะของไฟทั้งหมดในตาราง
2. getButton() : คืนค่าปุ่มที่ถูกกดของ state นั้น ๆ
3. setButton() : กำหนดว่า state ถูกกดมาจากปุ่มใด
4. equals(Object i) และ hashCode() : Override มาเพื่อนิยามการเท่ากันของตัวแปร PuzzleState โดยที่เมื่อตัวแปร PuzzleState มี code เหมือนกัน แสดงว่า PuzzleState นั้น ๆ เป็นตัวเดียวกัน (state แต่ละ state จะมีค่า code เฉพาะของตัวเองเนื่องจาก state นั้นเป็น Bit string ที่มีลักษณะเหมือนเลขฐาน 2 ทำให้ ถ้าเป็น state ที่ต่างกัน เมื่อแปลงเป็นเลขฐาน 10 แล้ว ก็จะได้ค่าที่แตกต่างกันไปของแต่ละ state ด้วย)

4. Class lightSolver

```
public class lightSolver {
    private Graph<PuzzleState,DefaultEdge> G;
    private int size;
    private int area;
    private PuzzleState startState;
    private PuzzleState endState;
    private String problem;

    public lightSolver(String lb,int board_size){...13 lines }

    public ArrayList<Integer> Solve(){...40 lines }

    public void printLayout(){...20 lines }

    public void printBoard(String s) {...25 lines }

    public void printSolution(ArrayList<Integer> Solution){...29 lines }

    public PuzzleState toggledState(PuzzleState p, int index) {...22 lines }

    public char toggle(char a) {...5 lines }

}
```

Class lightSolver เป็น Class ที่ทำหน้าที่แก้ปัญหา Light out puzzle โดยภายใน class มีการใช้วิธีการจัดเก็บข้อมูลแบบ ArrayList เนื่องจาก ArrayList สามารถย่อและขยายได้อัตโนมัติตามข้อมูลที่มีอยู่ภายในโดยไม่ต้องแจ้งล่วงหน้า ทำให้มันยืดหยุ่นในการทำงานเพราะไม่ต้องกำหนดขนาดสูงสุดล่วงหน้าเหมือนกับ Array ปกติ และมีการใช้วิธีการจัดเก็บข้อมูลแบบ ArrayDeque ที่ Implement Deque Interface มาทำให้สามารถเข้าถึงข้อมูลได้ทั้งด้านบนและด้านล่างของข้อมูล และมีการเก็บข้อมูลแบบ First in , First out (FIFO) หรือ Last in, First out (LIFO)

โดยมีตัวแปรดังนี้

1. Graph G : เป็นกราฟประเภท Simple Directed Graph ซึ่งกำหนดให้ Vertex เป็น Puzzle State โดยแต่ละ Puzzle State จะมี Edge เชื่อมไปยัง Puzzle State ที่สามารถเกิดขึ้นได้จากการกดปุ่มใด ๆ บนตารางของ Puzzle State นั้น ๆ
2. int size : เก็บขนาดของตาราง Puzzle
3. int area : เก็บพื้นที่ของตาราง Puzzle ซึ่งมีค่าเท่ากับจำนวนไฟทั้งหมดบนตาราง
4. PuzzleState startState : เก็บ state เริ่มต้นของ Puzzle ที่ผู้ใช้ต้องการประมวลผล

5. PuzzleState endState : เก็บ state ที่ Puzzle ถูกประมวลผลแล้ว หรือก็คือ state เมื่อไฟทุกดวงดับหมด

6. String problem : เก็บสถานะของไฟทั้งหมดในตารางของ Puzzle ที่ต้องการจะแก้ไขรูปแบบของ Bit string

และมี method ที่สำคัญ ดังนี้

1. Solve() : ใช้ในการประมวลผลตาราง Puzzle
2. printLayout() : ใช้ในการแสดงผลเค้าโครงของตาราง Puzzle
3. printBoard(String s) : ใช้ในการแสดงผลตาราง Puzzle ณ state ต่าง ๆ ตาม Bit string ที่รับเข้ามา
4. printSolution(ArrayList<Integer> Solution) : ใช้ในการแสดงผลขั้นตอนในการประมวลผลตาราง Puzzle
5. toggleState(PuzzleState p, int index) : ใช้ในการเปลี่ยน state ของ PuzzleState p เมื่อทำการกดปุ่มที่ตำแหน่ง index ตามกฎของ Puzzle
6. toggle(char a) : ใช้ในการสลับสถานะของไฟแต่ละดวงให้กลายเป็นสถานะตรงข้ามจากเดิม นั่นคือ จากเปิดเป็นปิด และ จากปิดเป็นเปิด

5. Class light (Main Class)

```
import java.util.*;
import java.util.Random;

public class light {

    public static void main(String[] args) { ...109 lines }

}
```

Class light เป็น Class ที่เปรียบเสมือนกับ Main Class ของโปรแกรม โดยจะทำหน้าที่ในการรับค่าและตรวจสอบ Input ที่รับเข้ามา

อัลกอริทึมที่ใช้ในการแก้ปัญหา

ในการแก้ปัญหานี้เราเลือกใช้ **ArrayDeque** ในการเก็บข้อมูลของแต่ละ **Puzzle State** จากนั้นนำข้อมูลของแต่ละ **Puzzle State** ไปใช้ในการสร้างกราฟแบบ **Simple Directed Graph** ซึ่งกำหนดให้ Vertex เป็น **Puzzle State** โดยแต่ละ **Puzzle State** จะมี Edge เชื่อมไปยัง **Puzzle State** ที่สามารถเกิดขึ้นได้จากการกดปุ่มใด ๆ บนตารางของ **Puzzle State** นั้น ๆ และใช้อัลกอริทึม **Breadth First Search** ในการไล่ตรวจสอบ Vertex ต่าง ๆ ภายในกราฟ จากนั้นทำการหาเส้นทางที่สั้นที่สุดที่ใช้ในการปิดไฟทุกดวงที่ถูกเปิดขึ้น (จาก **Puzzle State** เริ่มต้นไปยัง **Puzzle State** ที่ไฟดับทุกดวง) เพื่อให้ได้จำนวน Move ที่มีค่าน้อยที่สุดในการแก้ปัญหา ซึ่งเลือกใช้อัลกอริทึม **Dijkstra** เนื่องจาก กราฟที่สร้างขึ้นนั้นมีจุดเริ่มต้นในการไล่ตรวจสอบ Vertex ต่าง ๆ เพียงจุดเดียว (Single-source shortest path) และกราฟที่เราใช้ในการเก็บข้อมูลของแต่ละ **Puzzle State** นั้นเป็น Simple directed graph ซึ่งไม่มี Negative edge แนวนอน สำหรับกราฟที่ไม่มี weight แล้วการใช้ **Dijkstra's Algorithm** จะทำงานเหมือนกับการใช้ **Breadth-first search**

โดยกำหนดให้

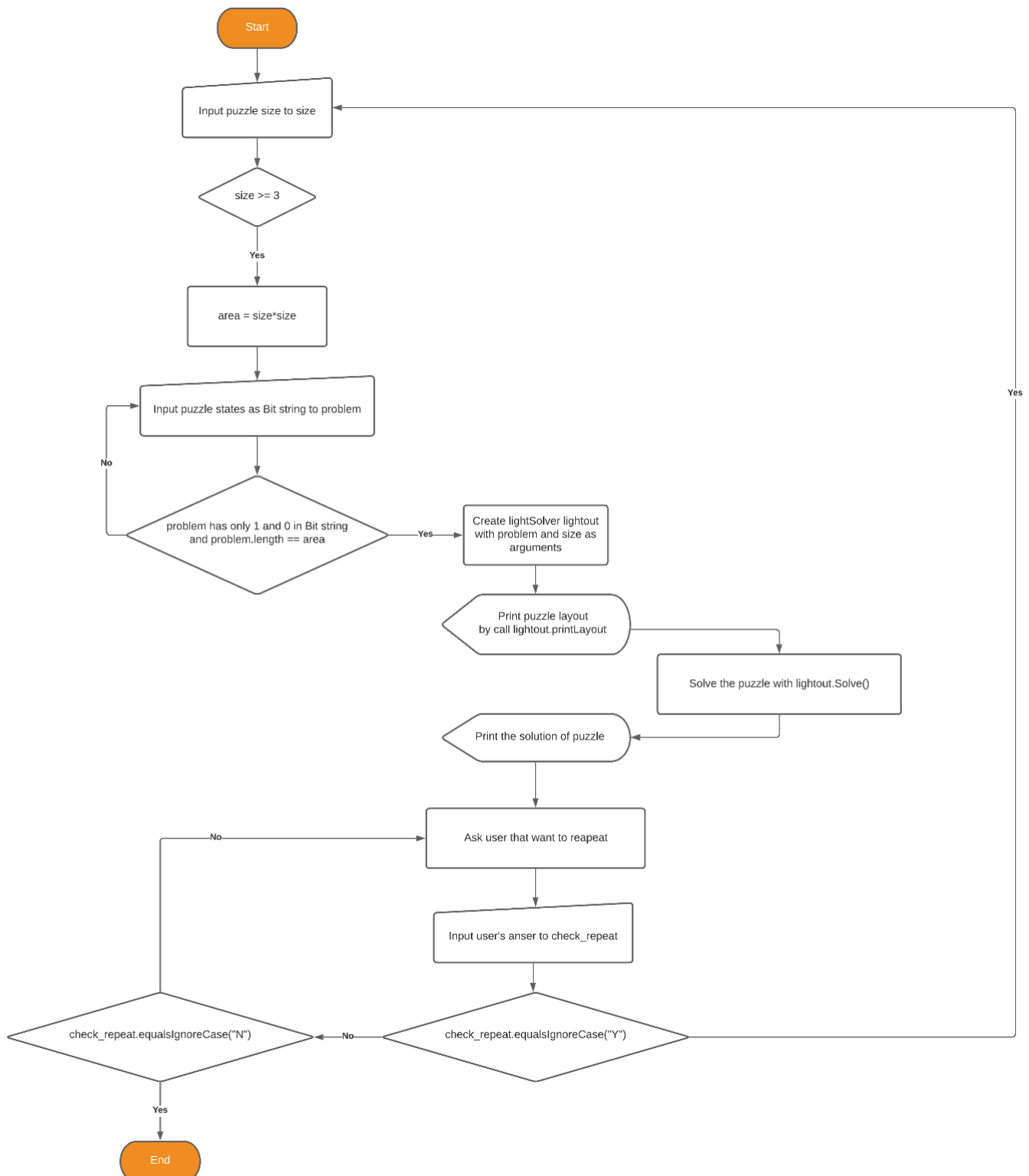
- puzzleQueue คือ ArrayDeque ของ **Puzzle State**
- graphPuzzle คือ กราฟของ **Puzzle State**
- startState คือ **Puzzle State** เริ่มต้น
- endState คือ **Puzzle State** ที่ไฟดับทุกดวง
- tempState คือ **Puzzle State** ตัวแรกใน puzzleQ
- newState คือ **Puzzle State** ที่สามารถเกิดขึ้นได้จากการกดปุ่มใด ๆ บน tempState
- solList คือ List ของ **Puzzle State** ของเส้นทางจาก startState ไปยัง endState
- solButton คือ ArrayList ของ ปุ่มที่ต้องกดเพื่อทำการดับไฟทุกดวง

ขั้นตอนในการประมวลผล

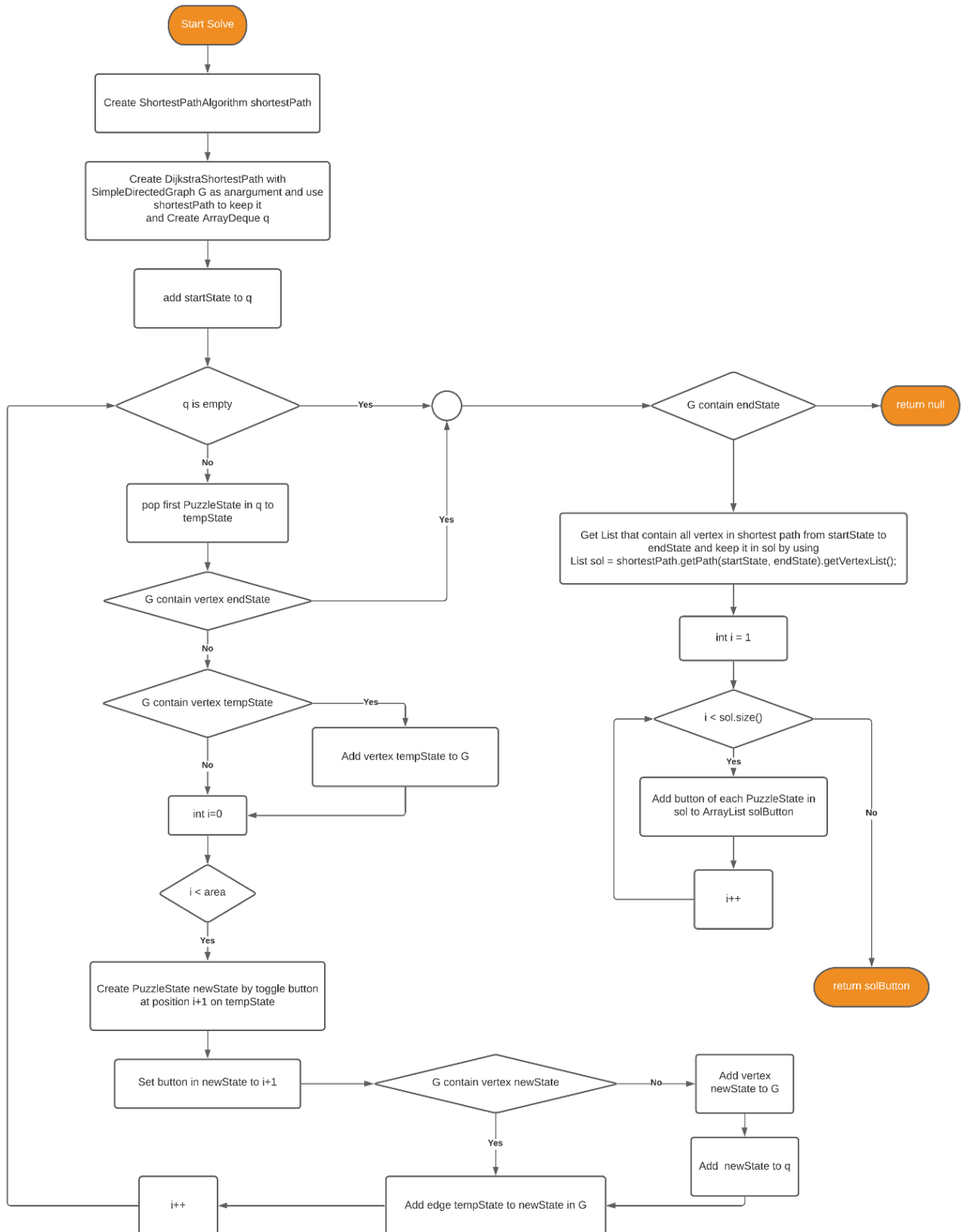
1. นำ Puzzle State เริ่มต้นไปใส่ใน puzzleQ และเริ่มวนลูปการประมวลผลจนกว่า puzzleQueue จะไม่มีข้อมูลใด ๆ เหลืออยู่
2. ให้ tempState คือ Puzzle State ตัวแรกที่ถูก pop ออกมาจาก puzzleQueue และนำไปตรวจสอบว่า tempState เท่ากับ endState หรือไม่ ถ้าใช่ให้ออกจากลูป puzzleQueue ถ้าไม่ใช่ให้ทำงานต่อไป จากนั้นให้ตรวจสอบว่า tempState นี้มีอยู่ในกราฟ graphPuzzle แล้วหรือยัง ถ้ายังให้เพิ่ม Vertex tempState ลงใน graphPuzzle
3. เริ่มการประมวลผลหา newState โดยให้ newState คือ Puzzle State ต่าง ๆ ที่เกิดขึ้นจากการกดปุ่มใด ๆ บน tempState ตั้งแต่ปุ่มแรกจนถึงปุ่มสุดท้าย และเก็บค่าว่า Puzzle State แต่ละอันนั้นเกิดมาจากการกดปุ่มใด แล้วนำไปตรวจสอบว่าในกราฟ graphPuzzle มี Vertex newState แล้วหรือยัง ถ้ายังให้ทำการเพิ่ม Vertex newState ลงไปในกราฟ graphPuzzle แล้วนำ newState ใส่ลงไปใน puzzleQueue
4. ทำการสร้าง Edge จาก tempState ไปยัง newState
5. ทำซ้ำข้อ 3-4 ไปเรื่อย ๆ จนครบทุกปุ่ม
6. ทำซ้ำข้อ 2-5 ไปเรื่อย ๆ จนกว่า puzzleQueue จะไม่มีข้อมูลใด ๆ เหลืออยู่ หรือออกจากลูปตั้งแต่เงื่อนไขข้อที่ 2 โดยขั้นตอนนี้เปรียบเสมือนการทำ Breath-first search ไปเรื่อย ๆ เพื่อทำการสร้างกราฟ graphPuzzle จนกว่าจะเจอ endState
7. เมื่อออกจากลูปแล้วให้ตรวจสอบว่ากราฟ graphPuzzle มี Vertex endState อยู่หรือไม่ถ้ามีแสดงว่า Puzzle นี้มีวิธีการดับไฟทุกดวง แต่ถ้าไม่มี แสดงว่า Puzzle นี้ไม่มีทางที่จะดับไฟทุกดวงได้
8. ถ้า Puzzle นี้มีวิธีการดับไฟทุกดวง ให้ทำการใช้อัลกอริทึม Dijkstra เพื่อทำการหาเส้นทางที่สั้นที่สุดจาก startState ไปยัง endState ภายในกราฟ graphPuzzle มาเก็บไว้ใน solList
9. เรียก Puzzle State แต่ละตัวภายใน solList (ยกเว้นตัวแรก เพราะตัวแรกจะเป็น startState) เพื่อทำการหาว่ามาจากการกดปุ่มใด แล้วนำมาเพิ่มลงไปใน solButton
10. จะได้วิธีในการดับไฟทุกดวงนั้นก็คือการกดปุ่มทุกปุ่มที่อยู่ใน solButton

แผนผังการทำงานของโปรแกรม (Flow Chart)

1. แผนผังการทำงานของ Main Function



2. แผนผังการทำงานของ Solve Function



อธิบายการทำงานของโปรแกรม

1. ส่วน Main class (Class light)

ทำหน้าที่หลักในการรับค่า Input แล้วทำการตรวจสอบ Input ว่าถูกต้องหรือไม่ จากนั้นทำการสร้าง Class lightSolver เพื่อทำการประมวลผลปัญหา

```
public class light {

    public static void main(String[] args) {
        int size = 0;
        int n = 0;
        int area = 0;
        String problem = null;
        boolean input_err = true;
        boolean repeat = true;
        Scanner input = null;

        while (repeat) {
            input_err = true;
            problem = null;
            System.out.println("=====Input puzzle data=====");
            while (input_err) {
                try {
                    System.out.println("Input puzzle size(N):");
                    input = new Scanner(System.in);
                    size = input.nextInt();
                    area = size * size;
                    if (size < 3) {
                        System.out.println("***** !!! WARNING !!! *****");
                        System.out.println("                Size of puzzle must be more than 3, please try again. ");
                        System.out.println("*****\n");
                    } else {
                        input_err = false;
                    }
                } catch (InputMismatchException e) {
                    System.out.println("***** !!! WARNING !!! *****");
                    System.out.println("                Type of input error (Integer only) or");
                    System.out.println("                A number is out of range(3,2147483647)");
                    System.out.println("                Please try again. ");
                    System.out.println("*****\n");
                    input_err = true;
                } catch (Exception NumberFormatException) {
                    System.out.println("***** !!! WARNING !!! *****");
                    System.out.println("                Type of input error (Integer only) or");
                    System.out.println("                A number is out of range(3,2147483647)");
                    System.out.println("                Please try again. ");
                    System.out.println("*****\n");
                    input.reset();
                }
            }
        }
    }
}
```

ประกาศค่าตัวแปร

loopเพื่อใช้ในการเริ่มรับ Input ใหม่เมื่อ User ต้องการประมวลผลอีกรอบ

รับค่าขนาดของตาราง Puzzle และตรวจสอบว่า format ของขนาด ถูกต้องหรือไม่ หากไม่ถูกต้องก็ให้ user ป้อน ขนาดใหม่อีกครั้ง

รับค่าสถานะของไฟทุกดวงเป็น Bit string และตรวจสอบว่า format ของ Bit string ถูกต้องหรือไม่ หากไม่ถูกต้องก็ให้ user ป้อน Bit string ใหม่อีกครั้ง

```
input_err = true;
while (input_err) {
    try {
        System.out.printf("Input puzzle states (%d bits, left to right, line by line):\n", area);
        input = new Scanner(System.in);
        problem = input.nextLine();
        boolean check = true;
        for (int i = 0; i < problem.length(); i++) {
            if (Integer.parseInt(String.valueOf(problem.charAt(i))) > 1) {
                System.out.println("Invalid state input!, please input only 0 or 1");
                System.out.println("***** !!! WARNING !!! *****");
                System.out.println("                Type of input error (Integer only) or");
                System.out.println("                A number of input must be only 0 or 1");
                System.out.println("                Please try again. ");
                System.out.println("*****\n");
                check = false;
                break;
            }
        }
        if (problem.length() == area && check) {
            input_err = false;
        } else if (check) {
            System.out.println("***** !!! WARNING !!! *****");
            System.out.println("                Invalid state input!, please input %d bits\n", area);
            System.out.println("*****\n");
        }
    } catch (NumberFormatException e) {
        System.out.println("Invalid state input!, please input only 0 or 1");
        System.out.println("***** !!! WARNING !!! *****");
        System.out.println("                Type of input error (Integer only) or");
        System.out.println("                A number of input must be only 0 or 1");
        System.out.println("                Please try again. ");
        System.out.println("*****\n");
        input_err = true;
    } catch (StringIndexOutOfBoundsException e) {
        System.out.println("***** !!! WARNING !!! *****");
        System.out.printf("                Invalid state input!, please input %d bits\n", area);
        System.out.println("*****\n");
        input_err = true;
    }
}
```

โดยโปรแกรมกำหนดไว้ว่าขนาดของตาราง Puzzle ที่ถูกต้อง จะต้องเป็นตัวเลขที่มีค่าน้อย 3 และมีค่าไม่เกิน 2137483647 (เนื่องจากเกินของเขตที่ตัวแปร integer สามารถรับค่าได้) และสถานะของไฟทุกดวงเป็น Bit string โดยจะต้องเป็น String ที่ประกอบไปด้วยค่า 0 หรือ 1 เท่านั้น และ Bit string ต้องมีความยาวเท่ากับขนาดของตาราง Puzzle ยกกำลังสอง ($N \times N$) หรือก็คือพื้นที่ของตาราง Puzzle

เมื่อ Input ถูกต้องแล้วจะสร้างตัวแปร lightSolver lightout โดยส่ง argument เป็นสถานะของไฟทุกดวงและขนาดของตาราง Puzzle แล้วทำการแสดงผลเค้าโครงของตาราง Puzzle หลังจากนั้นให้แสดงตารางปัญหาของ Puzzle ที่รับเข้ามาแล้วทำการแก้ปัญหา และแสดงผลคำตอบด้วยการเรียก lightout.printSolution(lightout.Solve()

```
lightSolver lightout = new lightSolver(problem,size);
lightout.printLayout();
System.out.println("=====Puzzle board=====");
System.out.println("Puzzle problem: ");
lightout.printBoard(problem);
lightout.printSolution(lightout.Solve());

boolean check_repeat = true;
String rp;
while (check_repeat) {
    System.out.println("Do you want to repeat ? (Y/N)");
    input = new Scanner(System.in);
    rp = input.nextLine();
    if (rp.equalsIgnoreCase("Y")) {
        check_repeat = false;
    } else if (rp.equalsIgnoreCase("N")) {
        repeat = false;
        check_repeat = false;
    } else {
        System.out.println("Invalid input!, Y to repeat or N to quit program");
    }
}
}
```

เมื่อโปรแกรมทำการประมวลผลปัญหาที่ได้รับมาเสร็จสิ้นแล้ว ให้ถาม User ว่าต้องการใช้งานโปรแกรมอีกครั้งหรือไม่ ถ้าใช่ให้ User ป้อน Y/y แล้วโปรแกรมจะกลับไปรับ Input ใหม่ แต่ถ้าไม่ใช่ให้ User ป้อน N/n แล้วโปรแกรมจะจบการทำงาน ถ้า User ป้อนอย่างอื่นนอกเหนือจากนี้ โปรแกรมจะให้ User ป้อนใหม่อีกครั้ง

2. ส่วนการประมวลผลการแก้ปัญหา (Class lightSolver)

2.1 Function Solve()

```
public ArrayList<Integer> Solve(){
    ShortestPathAlgorithm<PuzzleState, DefaultEdge> shortestPath=null;
    shortestPath = new DijkstraShortestPath<>(G);
    List<PuzzleState> sol = new ArrayList<PuzzleState>();
    ArrayList<Integer> solButton = new ArrayList<Integer>();

    ArrayDeque<PuzzleState> q = new ArrayDeque<>();
    PuzzleState tempState=null;
    q.add(startState);
    while (!q.isEmpty()) {
        tempState=q.pop();
        if(G.containsVertex(endState)){
            break;
        }
        if (!G.containsVertex(tempState)) {
            G.addVertex(tempState);
        }

        for (int i = 0; i < area; i++) {
            PuzzleState newState = toggledState(tempState, i);
            newState.setButton(i);
            if (!G.containsVertex(newState)) {
                G.addVertex(newState);
                q.add(newState);
            }
            G.addEdge(tempState,newState);
        }
    }
    if(!G.containsVertex(endState)){
        return null;
    }
    else{
        sol = shortestPath.getPath(startState, endState).getVertexList();
        for(int i=1;i<sol.size();i++){
            solButton.add(sol.get(i).getButton());
        }
        return solButton;
    }
}
```

ประกาศตัวแปรที่จำเป็น โดยที่

shortestPath คือ ตัวแปรที่ใช้ในการหาเส้นทางที่สั้นที่สุดจาก startState ไปยัง endState

sol คือ ตัวแปรที่เก็บ PuzzleState ต่าง ๆ ที่ใช้ในการเดินจาก startState ไปยัง endState **solButton** คือ ตัวแปรที่เก็บปุ่มที่เราต้องกดเพื่อให้เกิด endState

นำ startState ใส่ใน q แล้ววนลูปเพื่อทำ BFS จนกว่าจะทำงานครบทุก PuzzleState ที่เป็นไปได้

สร้าง ArrayDeque<Puzzle> q เพื่อทำการเก็บ PuzzleState ที่จำใช้ในการทำ BFS และ PuzzleState tempState เพื่อใช้ในการรับค่าจาก q

วนลูปกดปุ่มจนกว่าจะครบทุกปุ่มในตาราง Puzzle ณ tempState นั้น ๆ

ตรวจสอบว่าทำ BFS จนเจอ endState หรือยังถ้าเจอแล้วให้ออกจากลูป ถ้ายังไม่เจอให้ตรวจสอบว่าในกราฟ G มี Vertex tempState หรือยังถ้ายังให้เพิ่ม Vertex tempState เข้าไป

สร้าง Edge จาก tempState ไปยัง newState

เมื่อทำการกดปุ่มแล้วจะได้ newState ให้ตรวจสอบว่า newState นี้มีอยู่ในกราฟ G หรือยัง ถ้ายังไม่มีให้เพิ่ม Vertex newState เข้าไป

ถ้าในกราฟ G ไม่มี endState แสดงว่า Puzzle นี้ไม่มีคำตอบ ให้คืนค่า null กลับไป

ถ้าหากว่ากราฟ G มี endState อยู่ แสดงว่า Puzzle นี้มีคำตอบ โดยให้ sol มารับ List ของ PuzzleState ที่ได้จากการหา Shortest path จาก startState ไปยัง endState หลังจากนั้นให้ให้ solButton มารับค่าจาก sol ว่าแต่ละ PuzzleState นั้นได้มาจากการกดปุ่มใด

ตัวอย่างแสดงการประมวลผล

ตัวอย่าง : ขนาดของตาราง Puzzle คือ 3×3 และ สถานะเริ่มต้นของไฟทั้งหมดคือ 000110101

* หมายเหตุ : เขียนค่า state ของ PuzzleState แทนเพื่อความเข้าใจง่าย

1. โปรแกรมทำการรับค่าขนาดของตาราง puzzle จาก user ซึ่งมีค่าเท่ากับ 3 และทำการรับค่าสถานะเริ่มต้นของไฟทั้งหมดในรูปของ Bit String จาก user ซึ่งมีค่าเท่ากับ 000110101 จากนั้นทำการแสดงผลตาราง puzzle ใน state เริ่มต้นที่รับค่าเข้ามาดังภาพ

```

=====Input puzzle data=====
Input puzzle size(N):
3
Input puzzle states (9 bits, left to right, line by line):
000110101
=====Puzzle button layout=====

| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

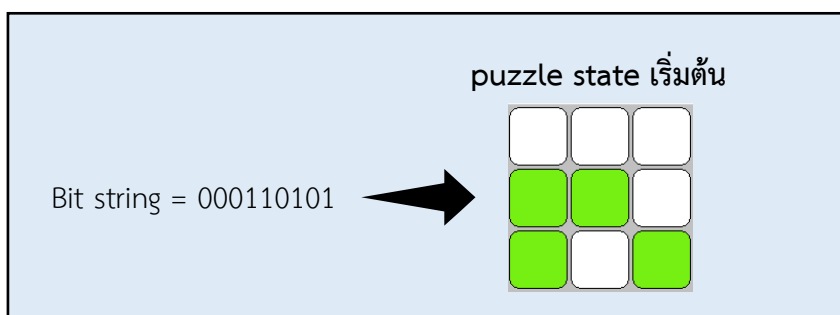
=====Puzzle board=====
Puzzle problem:
Bit string = 000110101
| 0 | 0 | 0 |
| 1 | 1 | 0 |
| 1 | 0 | 1 |

```

Puzzle button layout



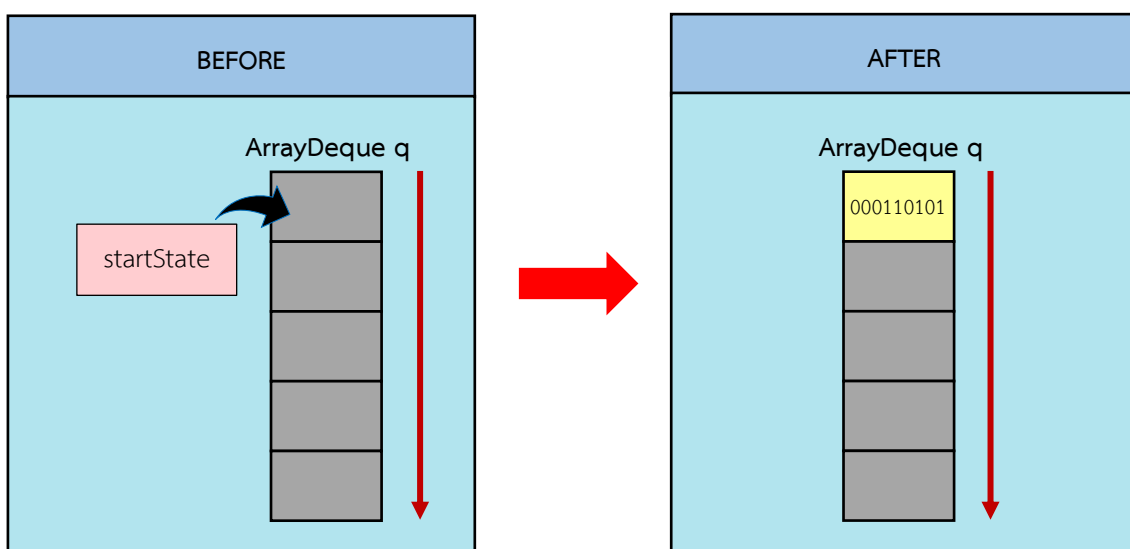
หมายเลขของไฟ ณ ตำแหน่งต่าง ๆ



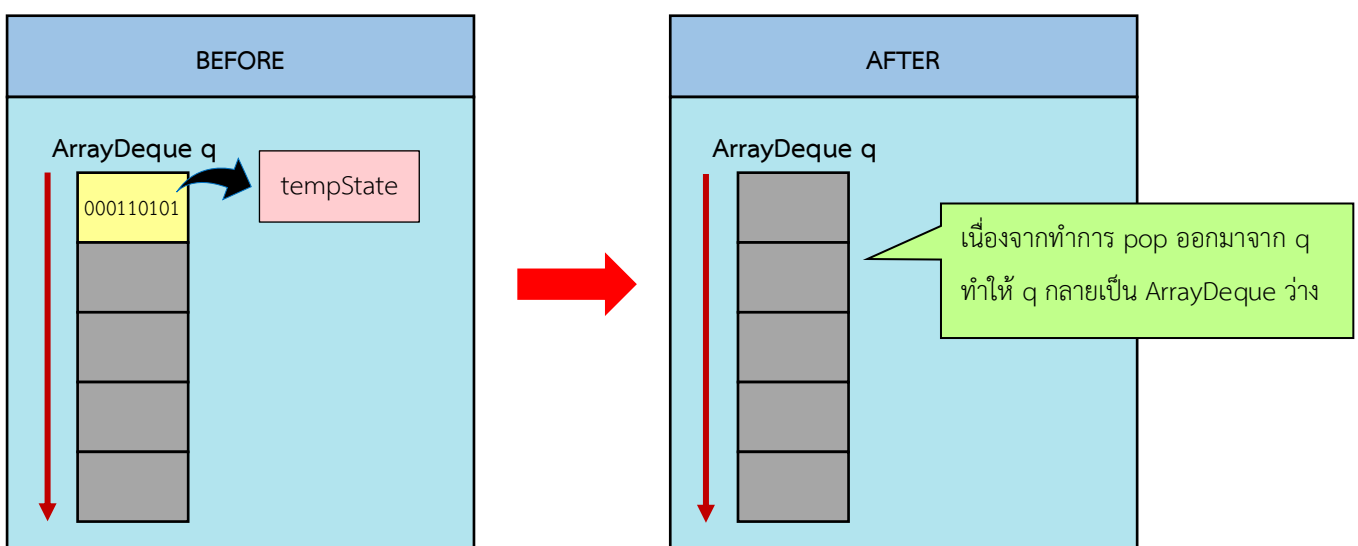
สัญลักษณ์ที่ใช้ในการอธิบาย

☒ : แทนไฟเปิด
☐ : แทนไฟปิด

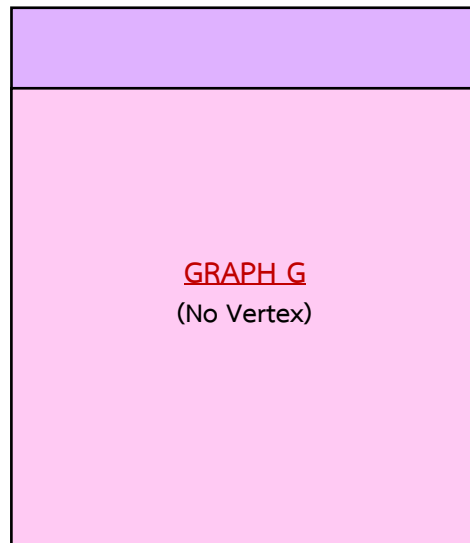
2. จากนั้นโปรแกรมจะเริ่มเข้าสู่ขั้นตอนการประมวลผลตาราง puzzle เพื่อคำนวณหาเส้นทางที่สั้นที่สุดที่ใช้ในการปิดไฟทุกดวงที่ถูกเปิดขึ้น โดยจะเข้าทำการประมวลผลผ่าน Function Solve() ใน class lightSolver
3. เริ่มต้นจากสร้างตัวแปร ShortestPathAlgorithm shortestPath เป็น DijkstraShortestPath
4. กำหนดให้ สถานะของไฟทั้งหมดตอนเริ่มต้น (startState) คือ 000110101 จากนั้นนำ startState เข้าไปเก็บไว้ใน q



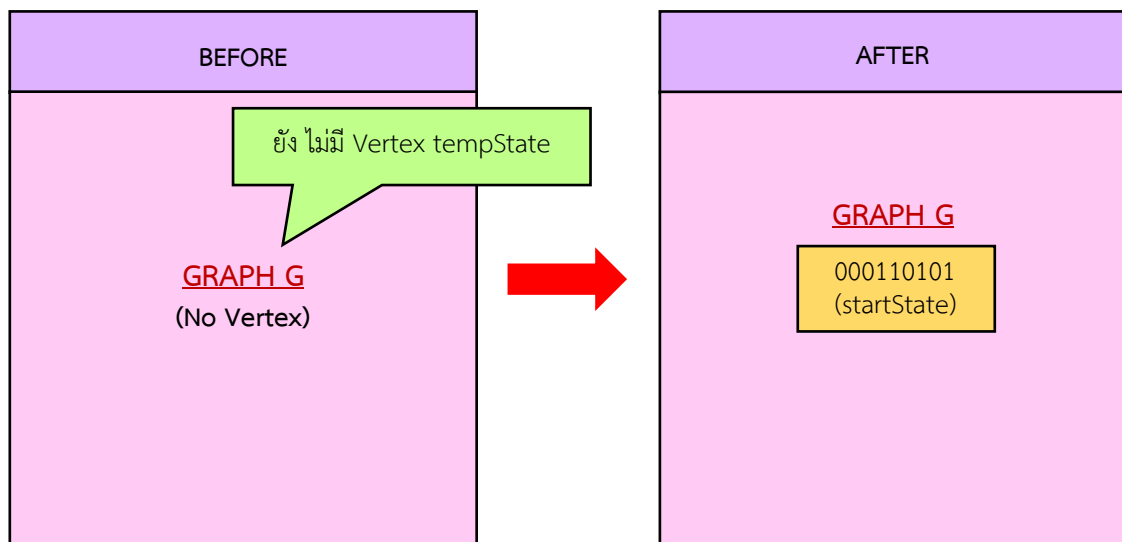
5. ตรวจสอบว่าปัจจุบัน *q* ว่างหรือไม่ ซึ่ง *q* ไม่ว่าง ดังนั้นจึงเข้าไปทำงานลูปต่อ
6. หลังจากนั้น เริ่มประมวลผลโดยการ pop PuzzleState ออกมาจาก *q* โดยนำมาเก็บไว้ใน tempState (ในรอบแรกจะ pop startState ออกมา)



7. ตรวจสอบว่าในกราฟ G มี endState (000000000) หรือยัง ซึ่งรอบแรกกราฟ G ยังไม่มี Vertex ใด ๆ ดังนั้นยังไม่มีดังนั้นไม่เข้าเงื่อนไข

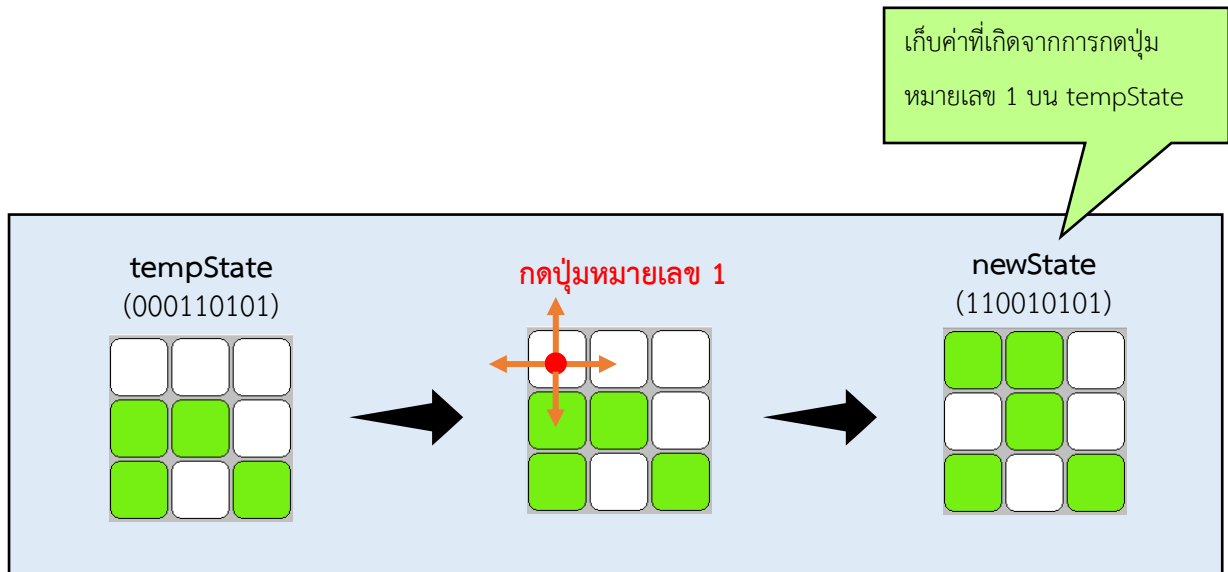


8. ตรวจสอบว่าในกราฟ G มี Vertex tempState (000110101) แล้วหรือยัง ซึ่งยังไม่มี ดังนั้นจึงเข้าเงื่อนไข โดยจะทำการเพิ่ม Vertex tempState ลงไปในกราฟ G

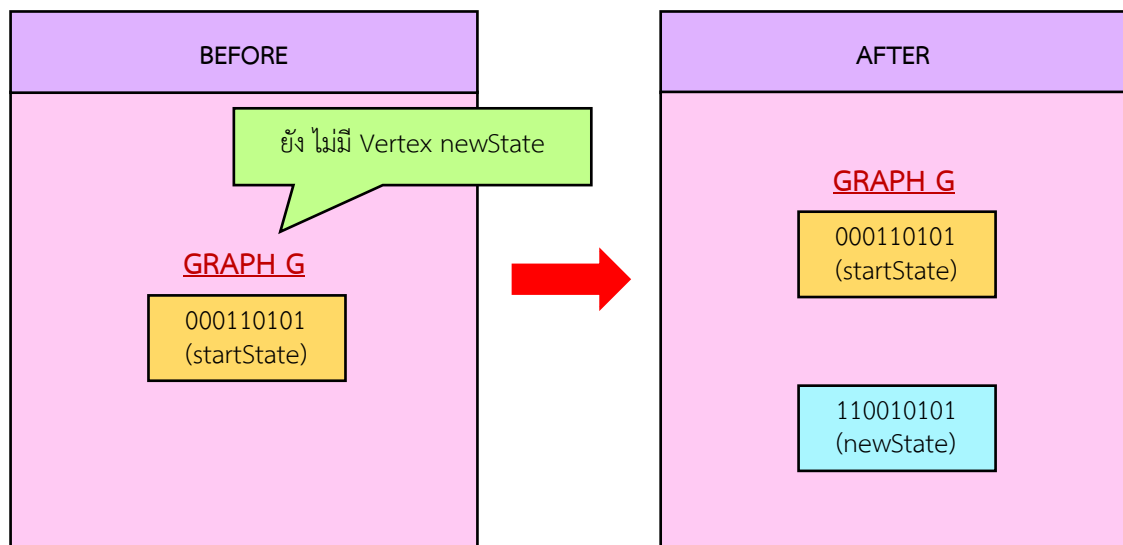


9. วนลูปเพื่อทำการกดปุ่มทุกปุ่มใน tempState ซึ่งจะเริ่มต้นจากการกดปุ่มหมายเลข 1

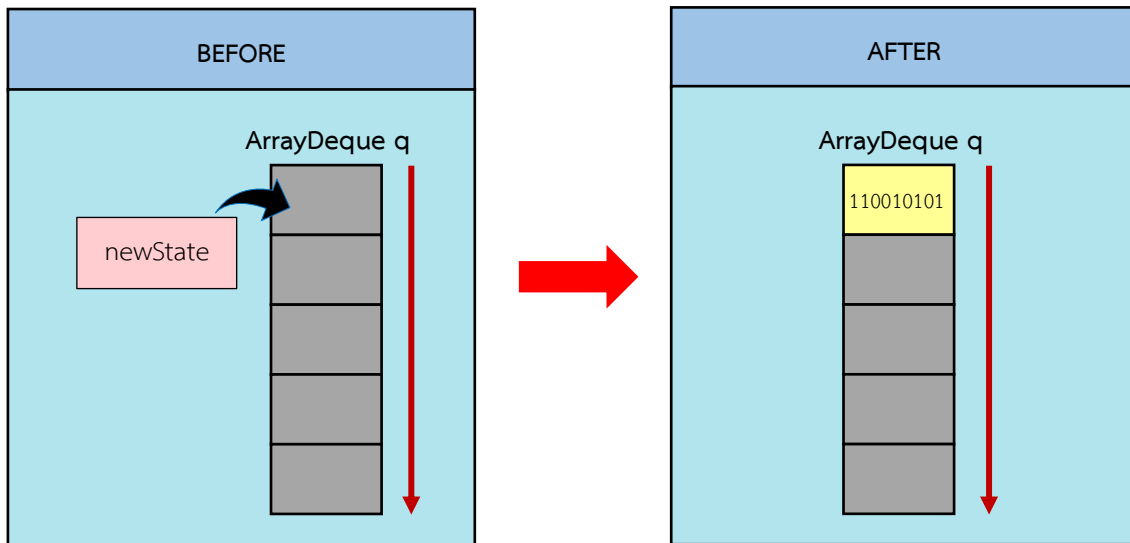
10. นำ PuzzleState ที่เกิดจากการกดปุ่มหมายเลข 1 มาเก็บไว้ที่ newState (110010101) และทำการ set ค่า button ของ newState ว่า state นี้เกิดมาจากการกดปุ่มใดบนตาราง Puzzle ซึ่งก็คือ ปุ่มหมายเลข 1



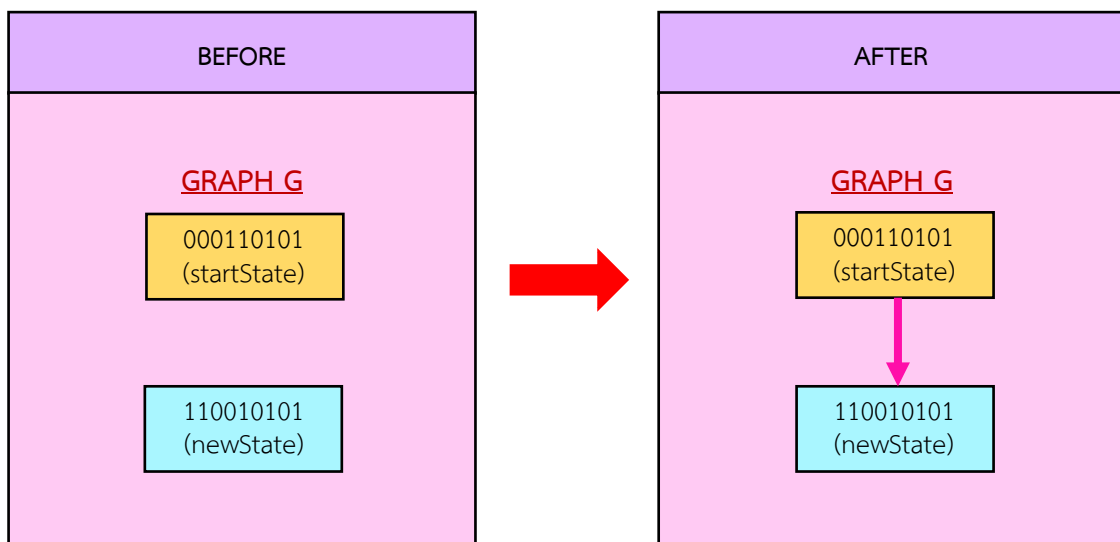
11. หลังจากสร้าง newState แล้วให้ทำการตรวจสอบว่าในกราฟ G มี Vertex newState แล้วหรือยัง ถ้ายังไม่มีให้ทำการเพิ่ม Vertex newState ลงไปในกราฟ G



12. จากนั้นทำการเพิ่ม newState ลงไปใน q แล้วจึงทำงานต่อ

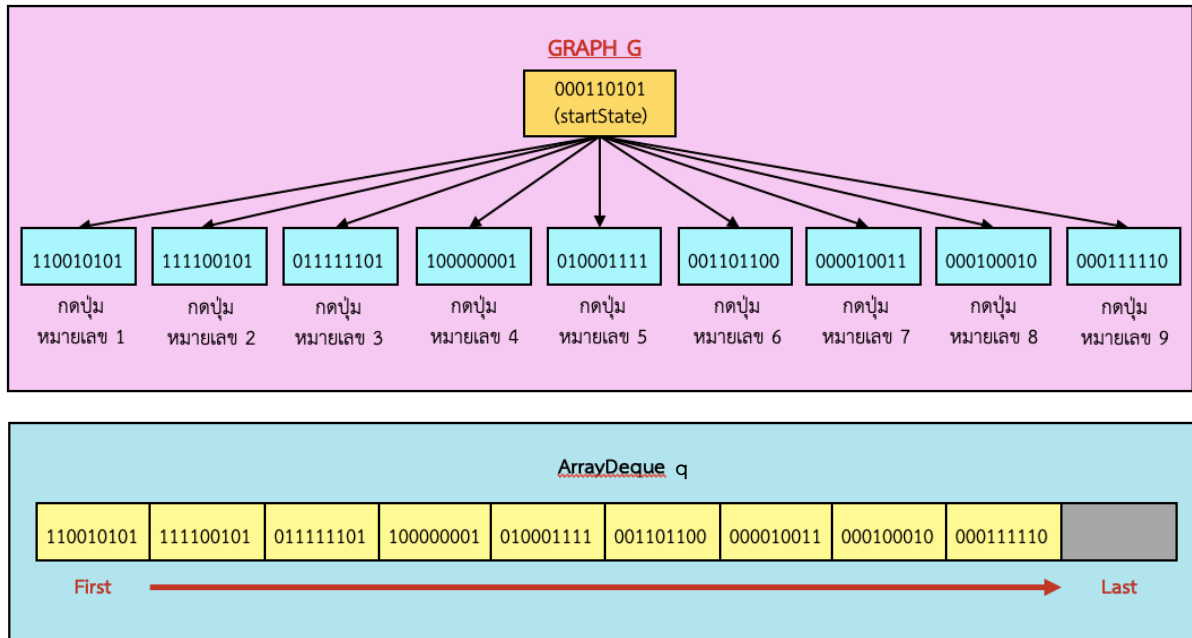


13. ทำการสร้าง Edge จาก tempState ไปยัง newState



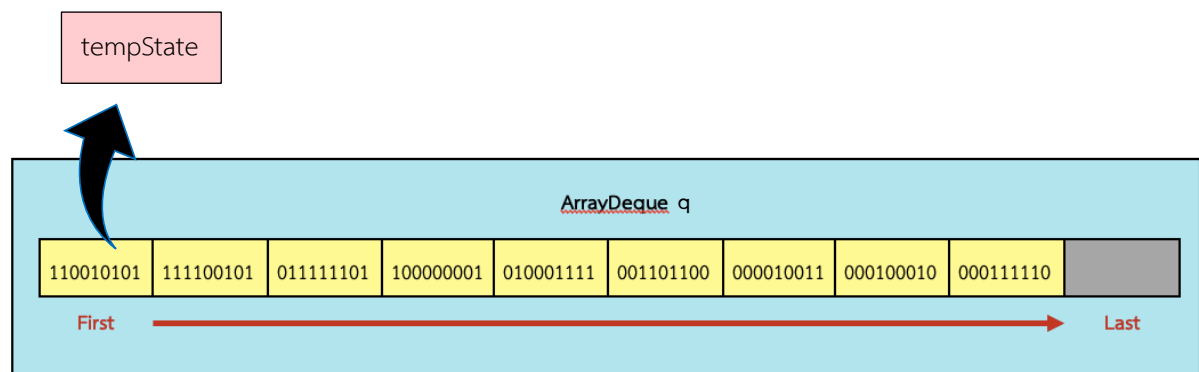
14. ทำไปเรื่อย ๆ จนกระทั่ง tempState วนจนกดปุ่มต่อไปจนครบทุกปุ่ม ตั้งแต่จากปุ่มหมายเลข 1 จนไปถึง ปุ่มหมายเลข 9

กราฟ G และ ArrayDeque q เมื่อ tempState ทำการกดปุ่มครบทุกปุ่ม

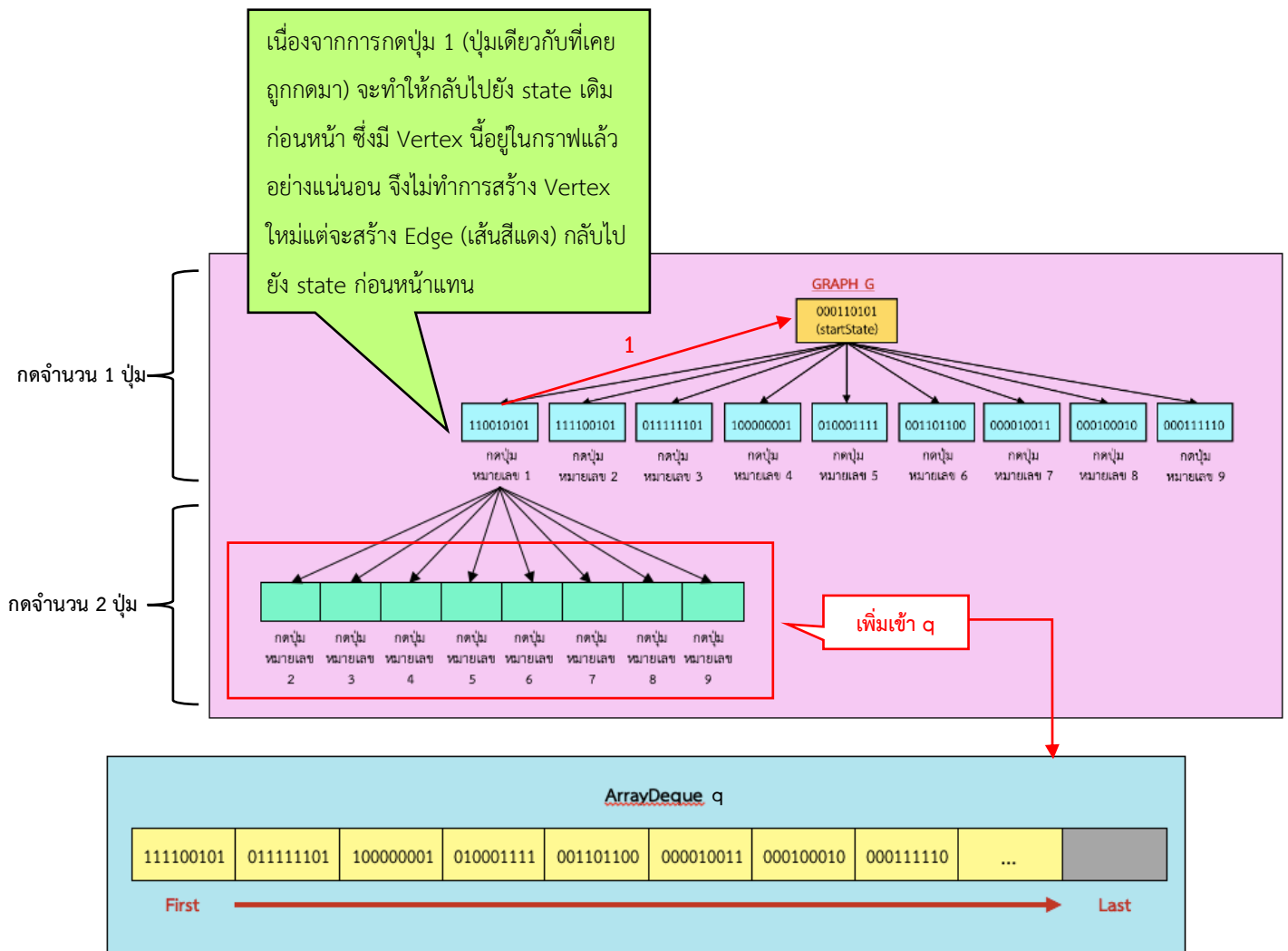


15. กลับไปเช็คค่า q ว่าว่างหรือไม่ ซึ่งไม่ว่าง ดังนั้นจึงเข้าสู่ loop ไปทำข้อ 3 - 10 ซ้ำจนกว่า q จะว่างหรือ เข้าเงื่อนไขที่ว่ากราฟ G มี endState อยู่จึงออกจาก loop

ตัวอย่างการทำประมวลผลรอบถัดไป



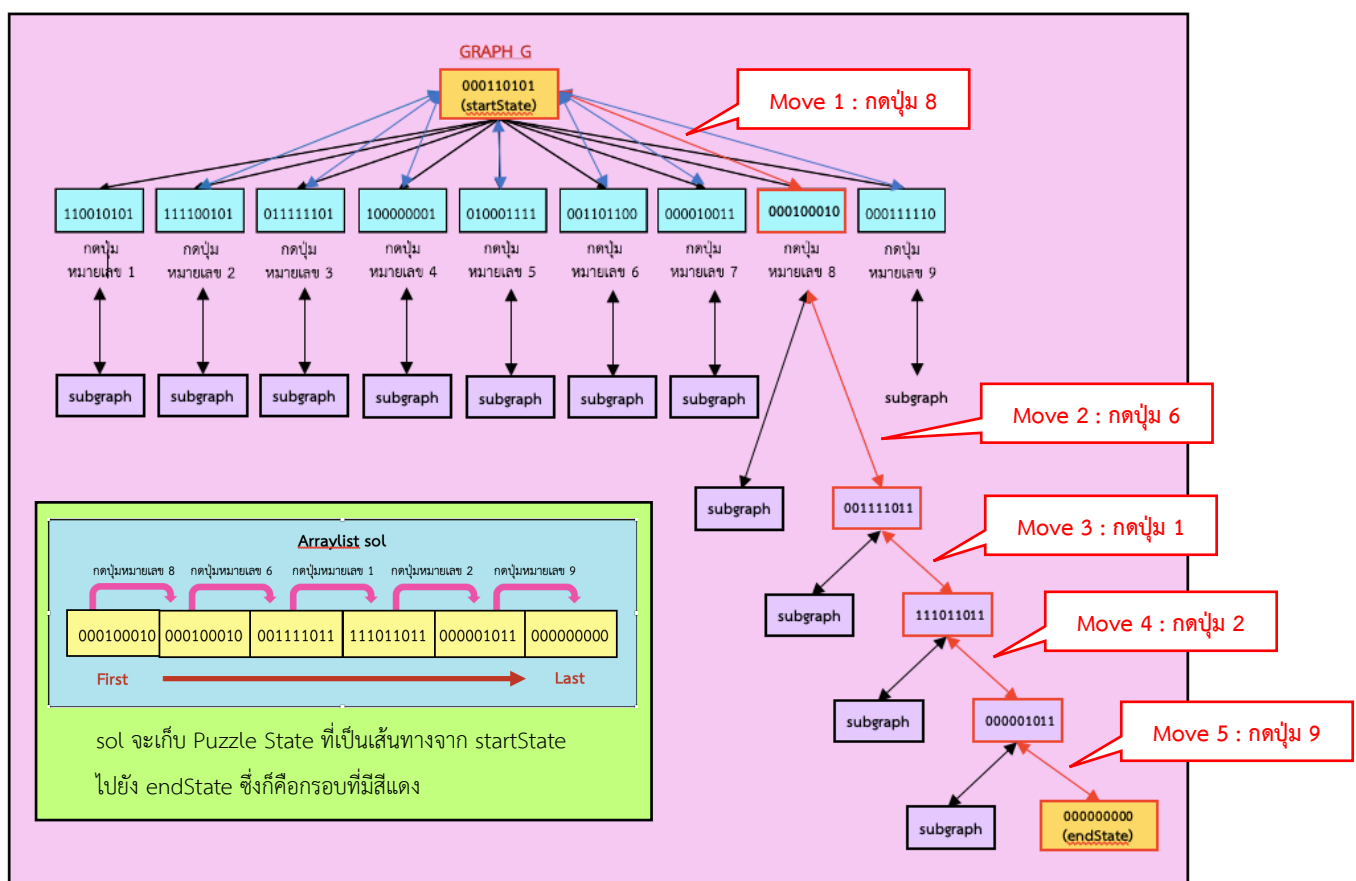
ทำซ้ำตั้งแต่ข้อ 3 - 10



16. จากสถานะเริ่มต้นของไฟทั้งหมดคือ 000110101 พบว่าจะวนลูปจนเจอ endState ในกราฟ G แล้วออกจากลูป ดังนั้นในกรณีนี้เป็น puzzle state ที่มี solution

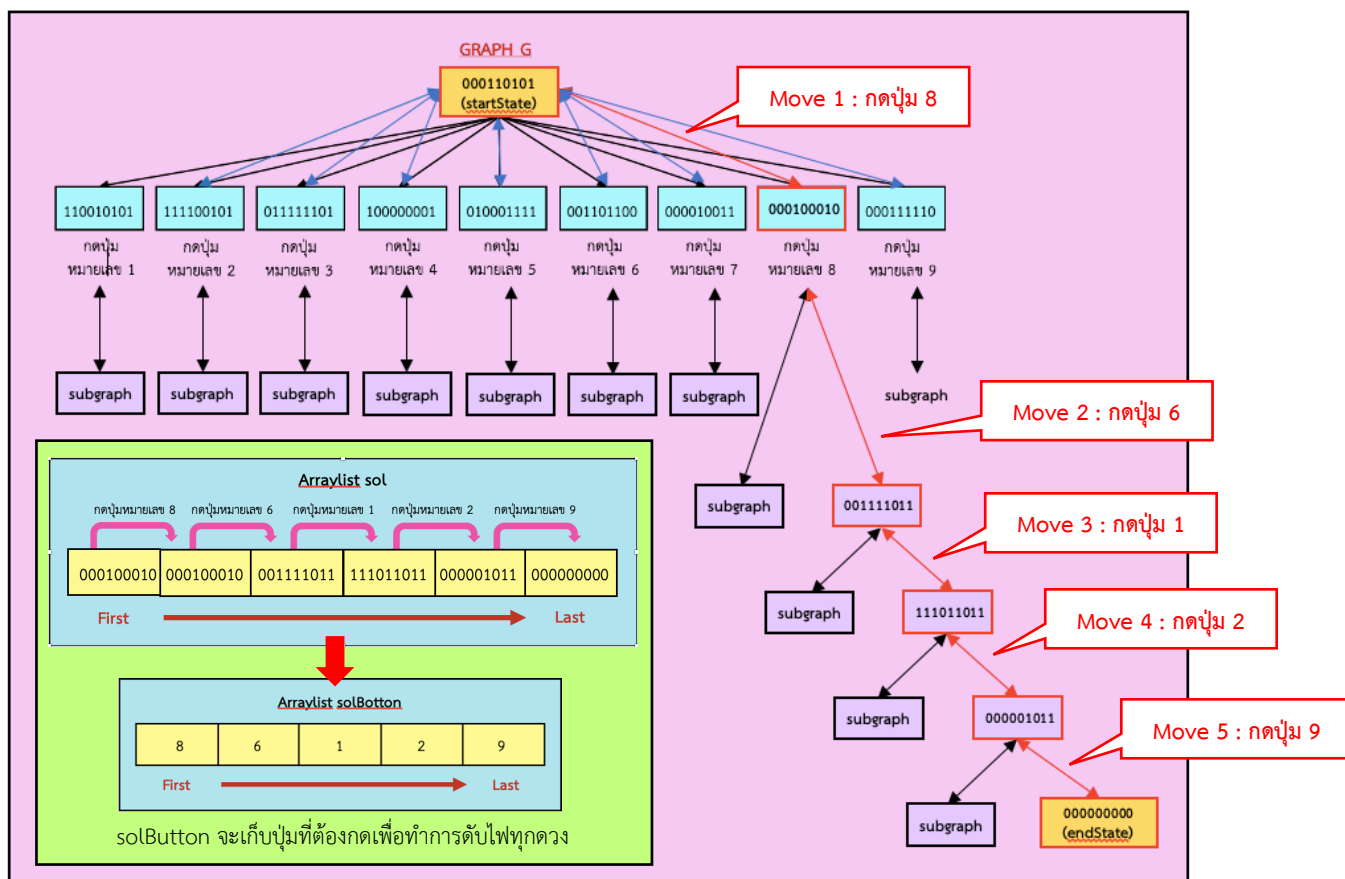
17. จากนั้นตรวจสอบว่า ถ้ากราฟ G ไม่มี endState ให้ return null ซึ่ง Input นี้ออกจากลูปมา ด้วยการเข้าเงื่อนไขเจอ endState ในกราฟ G ดังนั้นจึงไม่เข้าเงื่อนไขนี้ จึงเข้าไปทำงานใน else

18. ให้ shortestPath เรียกใช้ Function getPath(startState, endState) โดยส่ง Argument เป็น startState และ endState เพื่อทำการหาเส้นทางที่สั้นที่สุดจาก startState ไปยัง endState และเรียก Function getVertexList() เพื่อทำการคืนค่า List ที่บรรจุ PuzzleState แต่ละตัวที่เป็นเส้นทางที่สั้นที่สุดจาก startState ไปยัง endState มาเก็บไว้ที่ List sol

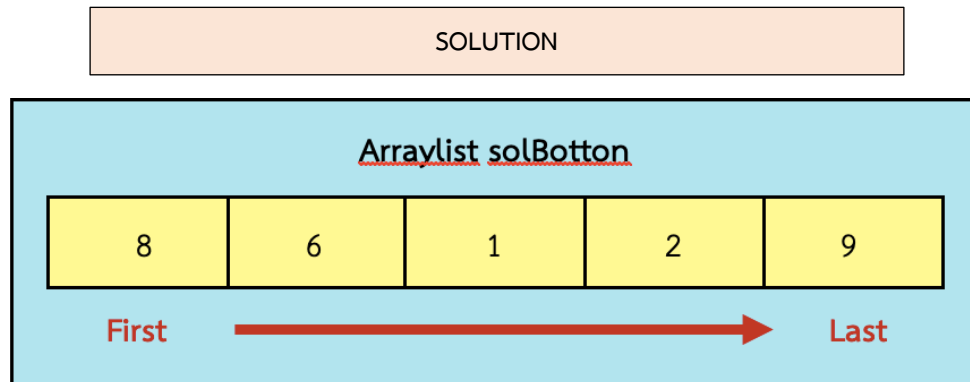


19. วนลูปเพื่อให้ PuzzleState ทุกตัวใน sol (ยกเว้นตัวแรก เนื่องจากเป็น startState) เรียกใช้ Function getButton() เพื่อดูว่า PuzzleState นี้เกิดมาจากการกดปุ่มใดบ้าง แล้วนำ list ของปุ่มที่ต้องกดเพื่อทำการดับไฟทุกดวง มาเพิ่มใน ArrayList solButton

Shortest path จาก startState ไปยัง endState

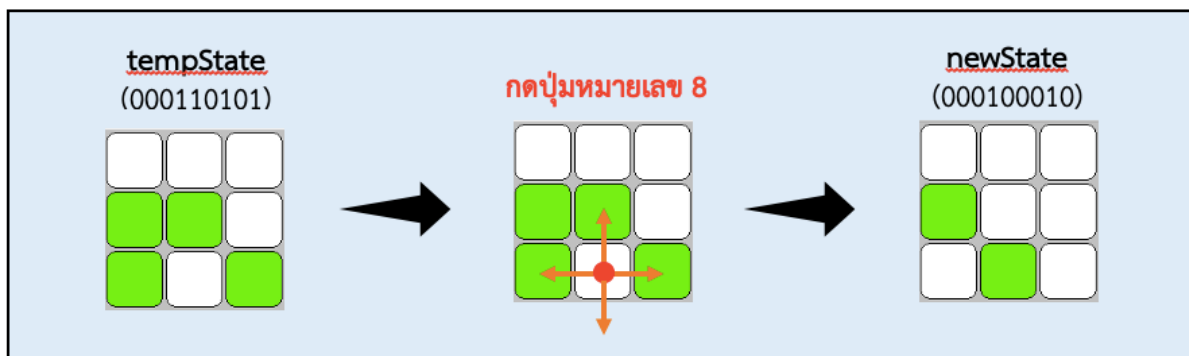


20. เมื่อออกจากลูปจะทำให้ solButton เก็บตำแหน่งปุ่มที่ต้องกดเพื่อทำการดับไฟทุกดวง หลังจากนั้นก็คืนค่า solButton ซึ่งเป็นวิธีในการดับไฟทุกดวงกลับไป เพื่อทำการแสดงผลขั้นตอนในการประมวลผลตาราง Puzzle



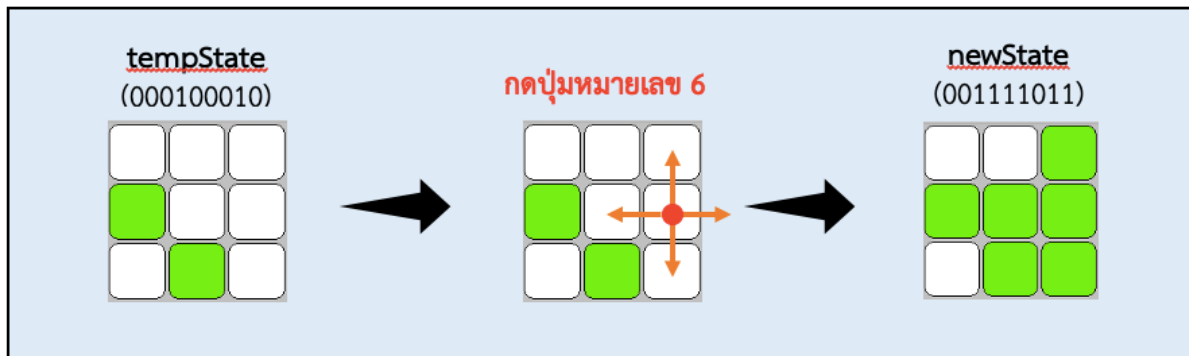
แสดงผลขั้นตอนในการประมวลผล

การ move ครั้งที่ 1



```
=====Puzzle solution=====
Puzzle solution: 8->6->1->2->9
5 moves to turn off all lights
Show solution:
>>> Move 1 : turn on button 8
Bit string = 000100010
-----
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
-----
```

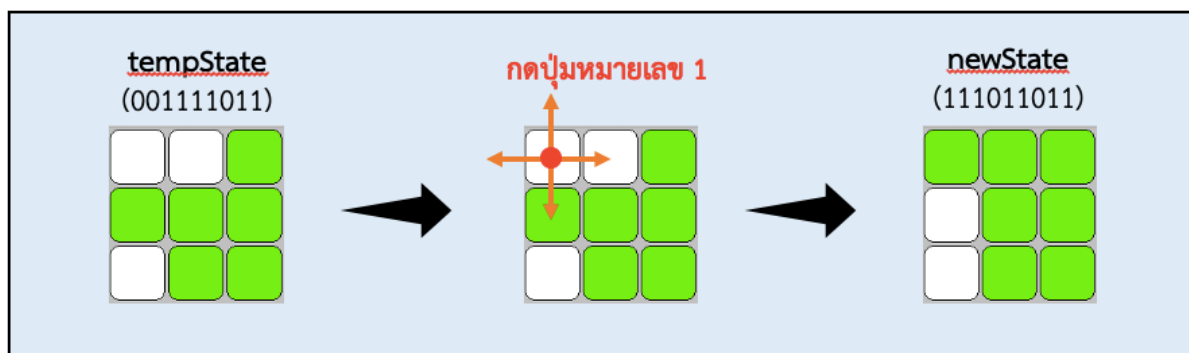
การ move ครั้งที่ 2



```
>>> Move 2 : turn on button 6
Bit string = 001111011
```

```
-----
| 0 | 0 | 1 |
-----
| 1 | 1 | 1 |
-----
| 0 | 1 | 1 |
-----
```

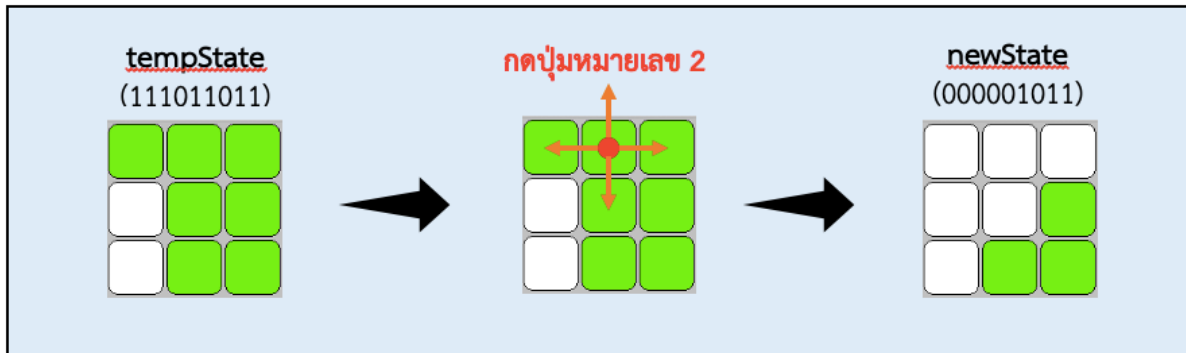
การ move ครั้งที่ 3



```
>>> Move 3 : turn on button 1
Bit string = 111011011
```

```
-----
| 1 | 1 | 1 |
-----
| 0 | 1 | 1 |
-----
| 0 | 1 | 1 |
-----
```

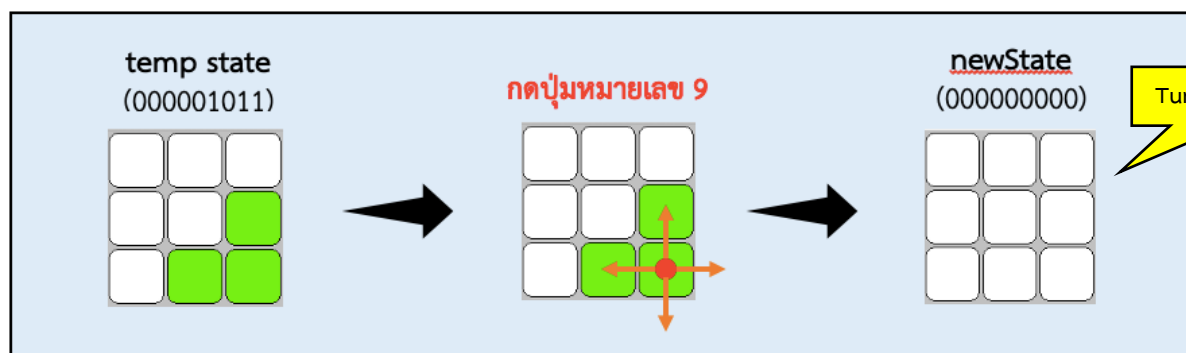
การ move ครั้งที่ 4



```
>>> Move 4 : turn off button 2
Bit string = 000001011
```

```
-----
| 0 | 0 | 0 |
-----
| 0 | 0 | 1 |
-----
| 0 | 1 | 1 |
-----
```

การ move ครั้งที่ 5



```
>>> Move 5 : turn off button 9
Bit string = 000000000
```

```
-----
| 0 | 0 | 0 |
-----
| 0 | 0 | 0 |
-----
| 0 | 0 | 0 |
-----
```

```
=====The puzzle has been solved!=====
Do you want to repeat ? (Y/N)
```


Runtime ของโปรแกรมนี้

```

public ArrayList<Integer> Solve(){
    ShortestPathAlgorithm<PuzzleState, DefaultEdge> shortestPath=null;
    shortestPath = new DijkstraShortestPath<>(G);
    List<PuzzleState> sol = new ArrayList<PuzzleState>();
    ArrayList<Integer> solButton = new ArrayList<Integer>();

    ArrayDeque<PuzzleState> q = new ArrayDeque<>();
    PuzzleState tempState=null;
    q.add(startState);
    while (!q.isEmpty()) {
        tempState=q.pop();
        if(G.containsVertex(endState)){
            break;
        }
        if (!G.containsVertex(tempState)) {
            G.addVertex(tempState);
        }

        for (int i = 0; i < area; i++) {
            PuzzleState newState = toggledState(tempState, i);
            newState.setButton(i);
            if (!G.containsVertex(newState)) {
                G.addVertex(newState);
                q.add(newState);
            }
            G.addEdge(tempState, newState);
        }
    }
    if(!G.containsVertex(endState)){
        return null;
    }
    else{
        sol = shortestPath.getPath(startState, endState).getVertexList();
        for(int i=1;i<sol.size();i++){
            solButton.add(sol.get(i).getButton());
        }
        return solButton;
    }
}

```

Asymptotic runtime ของลูปชั้นนอกในกรณีที่แย่ที่สุดคือ
เราต้องทำการกดปุ่มทุกปุ่มที่เป็นไปได้

ดังนั้นลูปนี้มี Asymptotic runtime = $O(2^n)$

เมื่อ n คือ จำนวนปุ่มทั้งหมดบนตาราง Puzzle

ในกรณีที่ไม่มีคำตอบ(กรณีที่แย่ที่สุด)
โปรแกรมนี้จะมี
Asymptotic runtime = $O(2^n)$

Asymptotic runtime ของการหา
Shortest Path ของอัลกอริทึม Dijkstra จะมี
Asymptotic runtime = $O((|V|+|E|) \log |V|)$
เมื่อ V คือจำนวน Vertex ใน G
และ E คือ จำนวน Edge ใน G

เราจะหา Asymptotic runtime จากกรณีที่แย่ที่สุดนั่นก็คือกรณีที่ไม่มีคำตอบซึ่งก็คือเราได้ทำการกดปุ่มทุกปุ่มที่สามารถเป็นไปได้แล้วไม่เจอ endState ในกราฟ G ทำให้ Asymptotic runtime ของลูปชั้นนอกหรือลูป while(!q.isEmpty()) จะมี Asymptotic runtime เท่ากับผลรวมของ nCr เมื่อ n คือจำนวนปุ่มทั้งหมดบนตาราง Puzzle และ r คือค่าตั้งแต่ 0 ถึง n ดังนั้นทั้งหมด จะมีค่าเท่ากับ $O(2^n)$ และเมื่อออกจากลูปมาแล้ว ถ้าไม่มีคำตอบโปรแกรมนี้จะ return null ทันที ดังนั้น โปรแกรมนี้จะมี Asymptotic runtime = $O(2^n)$

Algorithm นี้ถึงดีกว่าการ Brute-force อย่างไร?

Brute-force เป็นการทำงานแบบลองผิดลองถูก จึงทำให้บาง process เกิดการทำงานซ้ำได้ ซึ่งในกรณีนี้ Brute-force จะทำการกดปุ่มที่ปุ่มที่สามารถเป็นไปได้เพื่อหาคำตอบ ซึ่งจะมีค่า Asymptotic runtime = $O(2^n)$ ซึ่งจะเห็นว่ามี Asymptotic runtime เท่ากันกับ Algorithm ที่เราใช้ แต่ **Algorithm ที่เราใช้นี้** ถ้าหาก Puzzle นั้นมีคำตอบ จะทำให้เราสามารถหาจำนวนปุ่มที่น้อยที่สุดที่มี Runtime น้อยกว่าการ Brute-force อย่างไรก็ตามปัญหา Light out puzzle นี้ สามารถแก้ปัญหได้ด้วยอีกวิธีหนึ่งนั่นคือการใช้วิธีการใช้ Gaussian Elimination Algorithm เพื่อทำการแก้สมการหาว่าเราต้องกดปุ่มใดบ้างเพื่อให้ไฟทุกดวงดับซึ่งมี Asymptotic runtime = $O(m^3)$ เมื่อ m คือขนาดของตาราง Puzzle ซึ่งจะเห็นได้ว่าการใช้วิธี Gaussian Elimination Algorithm ที่มี Runtime เป็น Polynomial term จะมีประสิทธิภาพที่ดีกว่าการใช้ Shortest path Algorithm ที่มี Runtime เป็น Exponential term

ข้อจำกัดของโปรแกรม

เราจะทำการหาข้อจำกัดของโปรแกรมด้วยการหาว่าขนาดของ Puzzle เท่าใดจึงจะทำให้โปรแกรมเกิดปัญหา OutOfMemoryError ซึ่งจากการค้นคว้าและทดลองนั้นพบว่าโปรแกรมนี้จะเริ่มเกิดปัญหา OutOfMemoryError ที่ขนาด 5 x 5 และเมื่อมีขนาด 6 x 6 จะทำให้โปรแกรมไม่สามารถคำนวณเลขฐาน 2 ได้และเกิดปัญหา NumberFormatException ดังนั้น จึงเริ่มต้นการหาข้อจำกัดว่าที่ Puzzle ขนาด 5 x 5 จะไม่เกิดปัญหา OutOfMemoryError ที่คำตอบของ Puzzle ที่มีจำนวน move อย่างน้อยเท่าใด

1. Puzzle ขนาด 5 x 5

1.1 Bit String 0000001000111000100000000 → คำตอบ : 1 move

```
=====Puzzle board=====
Puzzle problem:
Bit string = 0000001000111000100000000
-----
| 0 | 0 | 0 | 0 | 0 |
-----
| 0 | 1 | 0 | 0 | 0 |
-----
| 1 | 1 | 1 | 0 | 0 |
-----
| 0 | 1 | 0 | 0 | 0 |
-----
| 0 | 0 | 0 | 0 | 0 |
-----
=====Puzzle solution=====
Puzzle solution: 12
1 moves to turn off all lights
```

1.2 Bit String 0000000100001101100001000 → คำตอบ : 2 move

```
=====Puzzle board=====
Puzzle problem:
Bit string = 0000000100001101100001000
-----
| 0 | 0 | 0 | 0 | 0 |
-----
| 0 | 0 | 1 | 0 | 0 |
-----
| 0 | 0 | 1 | 1 | 0 |
-----
| 1 | 1 | 0 | 0 | 0 |
-----
| 0 | 1 | 0 | 0 | 0 |
-----
=====Puzzle solution=====
Puzzle solution: 13->17
2 moves to turn off all lights
```

1.3 Bit String 0000001000111000100000000 → คำตอบ : 3 move

```

=====Puzzle board=====
Puzzle problem:
Bit string = 0000001000111110110101110
-----
| 0 | 0 | 0 | 0 | 0 | 0 |
-----
| 0 | 1 | 0 | 0 | 1 | 1 |
-----
| 1 | 1 | 1 | 1 | 1 | 1 |
-----
| 0 | 1 | 1 | 0 | 1 | 1 |
-----
| 0 | 1 | 1 | 1 | 0 | 1 |
-----
=====Puzzle solution=====
Puzzle solution: 12->15->23
3 moves to turn off all lights

```

1.4 Bit String 0100001101100111100111100 → คำตอบ : 4 move

```

=====Puzzle board=====
Puzzle problem:
Bit string = 0100001101100111100111100
-----
| 0 | 1 | 0 | 0 | 0 | 0 |
-----
| 0 | 1 | 1 | 0 | 1 | 1 |
-----
| 1 | 0 | 0 | 1 | 1 | 1 |
-----
| 1 | 1 | 0 | 0 | 1 | 1 |
-----
| 1 | 1 | 1 | 0 | 0 | 0 |
-----
=====Puzzle solution=====
Puzzle solution: 15->7->11->22
4 moves to turn off all lights

```

1.5 Bit String 0100111101111101100010111 → คำตอบ : 5 move

```

=====Puzzle board=====
Puzzle problem:
Bit string = 0100111101111101100010111
-----
| 0 | 1 | 0 | 0 | 1 | 1 |
-----
| 1 | 1 | 1 | 0 | 1 | 1 |
-----
| 1 | 1 | 1 | 1 | 0 | 1 |
-----
| 1 | 1 | 0 | 0 | 0 | 0 |
-----
| 1 | 0 | 1 | 1 | 1 | 1 |
-----
=====Puzzle solution=====
Puzzle solution: 7->16->10->14->24
5 moves to turn off all lights

```

1.6 Bit String 111111011111010101111110 → คำตอบ : 6 move

```

=====Puzzle board=====
Puzzle problem:
Bit string = 111111011111010101111110
-----
| 1 | 1 | 1 | 1 | 1 |
-----
| 1 | 1 | 0 | 1 | 1 |
-----
| 1 | 1 | 1 | 0 | 1 |
-----
| 0 | 1 | 0 | 1 | 1 |
-----
| 1 | 1 | 1 | 1 | 0 |
-----
=====Puzzle solution=====
Puzzle solution: 3->10->19->6->13->22
6 moves to turn off all lights

```

1.7 Bit String 00111101111101101110111 → คำตอบ : 7 move

```

=====Puzzle board=====
Puzzle problem:
Bit string = 00111101111101101110111
-----
| 0 | 0 | 1 | 1 | 1 |
-----
| 1 | 0 | 1 | 1 | 1 |
-----
| 1 | 1 | 1 | 0 | 1 |
-----
| 1 | 1 | 0 | 1 | 1 |
-----
| 1 | 0 | 1 | 1 | 1 |
-----
=====Puzzle solution=====
Puzzle solution: 11->17->21->15->4->13->24
7 moves to turn off all lights

```

1.8 Bit String 0000000011111010100101000 → คำตอบ : 8 move

```

=====Puzzle board=====
Puzzle problem:
Bit string = 0000000011111010100101000
-----
| 0 | 0 | 0 | 0 | 0 |
-----
| 0 | 0 | 0 | 1 | 1 |
-----
| 1 | 1 | 1 | 0 | 1 |
-----
| 0 | 1 | 0 | 0 | 1 |
-----
| 0 | 1 | 0 | 0 | 0 |
-----
=====Puzzle solution=====
Puzzle solution: 11->17->21->15->4->13->24
7 moves to turn off all lights

```

OutOfMemoryError

```

***** !!! WARNING !!! *****
Error! Out of memory
Can't find solution of the puzzle
*****

```

จะพบว่าเมื่อคำตอบของ Puzzle มี 8 move จะทำให้โปรแกรมนี้เกิดปัญหา OutOfMemoryError ทำให้เมื่อคำตอบของ Puzzle มีมากกว่า 8 move ก็จะทำให้เกิดปัญหา OutOfMemoryError เช่นเดียวกัน ดังนั้นข้อจำกัดของโปรแกรมนี้นี้คือ Puzzle ขนาดไม่เกิน 5 x 5 และมีจำนวน move ที่ต้องใช้ในการปิดไฟไม่เกินจำนวน 8 move

แหล่งอ้างอิง

1. What is the algorithm for solving lights out puzzle in minimum number of moves in Java? [ออนไลน์]. (วันที่ค้นหาข้อมูล : 27 เมษายน 2564). เข้าถึงได้จาก : <https://www.quora.com/What-is-the-algorithm-for-solving-lights-out-puzzle-in-minimum-number-of-moves-in-Java>
2. Lights Out Puzzle. [ออนไลน์]. (วันที่ค้นหาข้อมูล : 27 เมษายน 2564). เข้าถึงได้จาก : <https://mathworld.wolfram.com/LightsOutPuzzle.html>
3. Breadth First Search or BFS for a Graph. [ออนไลน์]. (วันที่ค้นหาข้อมูล : 27 เมษายน 2564). เข้าถึงได้จาก : <https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/>
4. เมื่อ Array มีข้อจำกัดจะทำยังไงดี? [ออนไลน์]. (วันที่ค้นหาข้อมูล : 29 เมษายน 2564). เข้าถึงได้จาก : <https://thefah-th.medium.com/เมื่อ-array-มีข้อจำกัดจะทำยังไงดี-4c0e1708160b>
5. Using Dijkstra's algorithm with negative edges? [ออนไลน์]. (วันที่ค้นหาข้อมูล : 29 เมษายน 2564). เข้าถึงได้จาก : <https://cs.stackexchange.com/questions/2482/using-dijkstras-algorithm-with-negative-edges>
6. Dijkstra's Algorithm complexity vs BFS complexity. [ออนไลน์]. (วันที่ค้นหาข้อมูล : 1 พฤษภาคม 2564). เข้าถึงได้จาก : <https://stackoverflow.com/questions/65767874/dijkstras-algorithm-complexity-vs-bfs-complexity>
7. LightsOut. [ออนไลน์]. (วันที่ค้นหาข้อมูล : 1 พฤษภาคม 2564). เข้าถึงได้จาก : <https://www.cs.ox.ac.uk/admissions/undergraduate/courses/lightsout.html>