Project Report

Light Out Puzzle

เสนอ

รศ.ดร.รังสิพรรณ มฤคทัต

จัดทำโดย

นายวัชรศักย์ พรหมณี	6413110
นายศศิศ ศรีรัตน์	6413112
นายกวิน เก่งเกตุ	6413210
นายรวิพล ฉายรักษา	6413223

รายงานนี้เป็นส่วนหนึ่งของวิชาโครงสร้างข้อมูลและขั้นตอนวิธี (วศคพ 221)

ภาคเรียนที่ 2 ปีการศึกษา 2565

คณะวิศวกรรมศาสตร์ สาขาวิชาวิศวกรรมคอมพิวเตอร์

มหาวิทยาลัยมหิดล

คำนำ

รายงานฉบับนี้เป็นส่วนหนึ่งของวิชา **โครงสร้างข้อมูลและขั้นตอนวิธี (วศคพ 221)** โดยจัดทำขึ้นเพื่อใช้อธิบายหลักการทำงานของโปรแกรมแก้ปัญหา Light Out Puzzle ซึ่งประกอบไปด้วยคู่มือการใช้งานโปรแกรมเบื้องต้น การอธิบายในส่วนของคำสั่งหรือโค้ด รูปแบบ Algorithm ที่เลือกใช้ รวมไปถึงข้อจำกัดต่าง ๆ ในการใช้งานโปรแกรม

ทางคณะผู้จัดทำขอขอบพระคุณอาจารย์ประจำวิชา รองศาสตราจารย์ ดร.รังสิพรรณ มฤคทัต ผู้ให้ความรู้ และแนวทางศึกษา จนโปรแกรมและรายงานฉบับนี้สามารถออกมาได้อย่างสมบูรณ์แบบ

สุดท้ายนี้ทางคณะผู้จัดทำหวังว่ารายงานฉบับนี้จะสามารถเป็นประโยชน์ไม่มากก็น้อยแก่ผู้อ่านทุกท่าน หากรายงานฉบับนี้เกิดข้อผิดพลาดหรือมีคำที่ตกหล่นคำใด ทางคณะผู้จัดทำต้องขออภัยมา ณ ที่นี้ด้วย

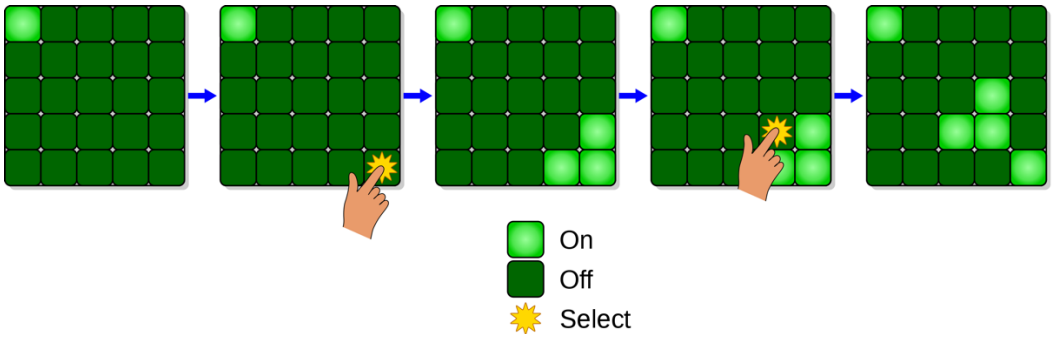
ขอขอบพระคุณ

คณะผู้จัดทำ

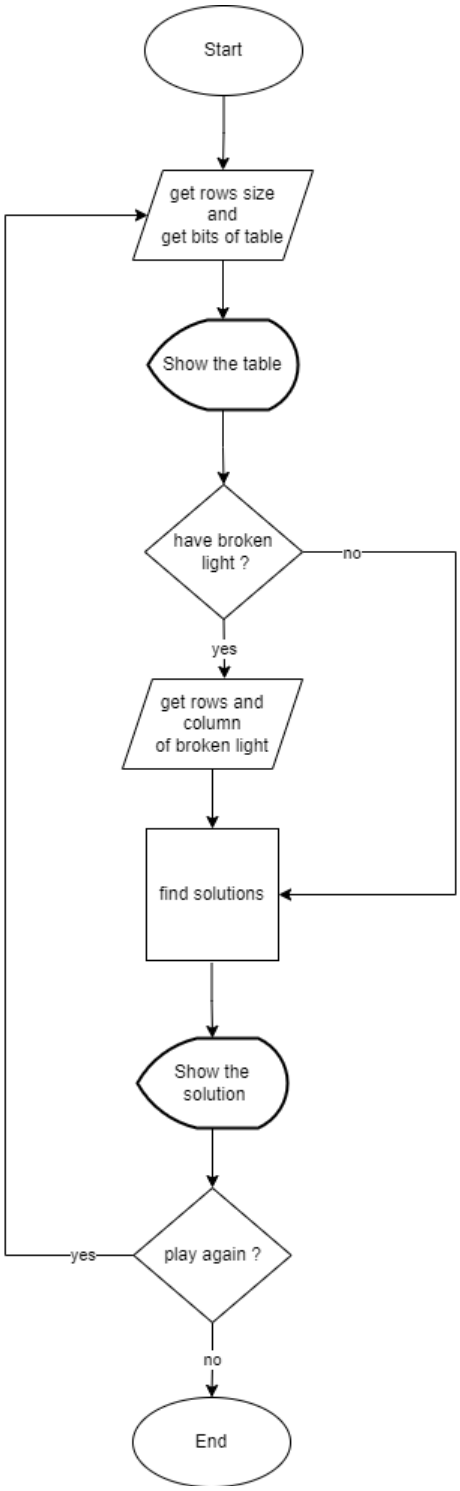
สารบัญ

เกี่ยวกับโปรแกรม

Light out puzzle คือ เกมพัฒนาทักษะทางคณิตศาสตร์และตรรกะที่มีการเรียงกล่องไฟให้ดับโดยการกดที่ทีละกล่อง โดยเมื่อกดแล้ว กล่องที่อยู่ในแถวและคอลัมน์เดียวกันกับกล่องที่กดจะเปลี่ยนสถานะเป็นกล่องดับ และกล่องที่อยู่ติดกับกล่องที่กดและมีสถานะเป็นกล่องติดจะเปลี่ยนสถานะเป็นกล่องติด เป้าหมายของเกมนี้คือการให้หลอดไฟให้ทั้งหมดเป็นสถานะปิด โดยจะต้องใช้จำนวนการกดที่น้อยที่สุด ทำให้ Light out puzzle เป็นเกมที่ท้าทายและเป็นที่นิยมกันในหลายๆ โครงการทางคณิตศาสตร์และวิทยาศาสตร์อื่น ๆ ด้วยเช่นกัน นอกจากนี้เกมนี้ยังมีชื่ออื่น ๆ อย่างเช่น **"Lights Off"** หรือ **"Switch Off"** และมีรูปแบบที่แตกต่างกันไปได้ในบางเวอร์ชั่นของเกมด้วย



Flow Chart ของโปรแกรม



คู่มือการใช้งานโปรแกรม

1. ป้อนจำนวนแถว (Rows) ที่ต้องการจะใส่ผ่าน Keyboard โดยจำนวนที่จะกรอกต้องเป็นเลขจำนวนเต็มเท่านั้น และต้องมีค่าไม่น้อยกว่า 2 หลังจากนั้นจะต้องป้อนกระดานเริ่มต้นเข้าไป ซึ่งจะอยู่ในรูปแบบบิต โดยมีค่าเท่ากับจำนวนแถวยกกำลัง 2 หลังจากนั้น โปรแกรมจะสร้างกระดานเริ่มต้นตามบิตที่ป้อนเข้ามา

```
Enter number of rows square grid =
2
Enter initial states (4 bits, left to right, line by line) =
1111
States in bits = 1111
       | col 0 | col 1
 row 0 |   1   |   1
 row 1 |   1   |   1

************************************************************
```

2. โปรแกรมจะถามว่าต้องการใส่หลอดไฟที่เสียหรือไม่ ซึ่งจะต้องตอบโดยการป้อน **Y** หรือ **N** ผ่าน Keyboard

2.1 หากป้อน **Y** หมายถึง **Yes** คือต้องการใส่หลอดไฟที่เสียเข้าไป หลอดไฟที่เสียจะมีตัว x กำกับอยู่ข้างหลัง โดยหลอดไฟเสียจะแตกต่างจากหลอดไฟปกติคือ เมื่อเปิดหรือปิดหลอดไฟที่เสีย โปรแกรมจะทำการเปิดหรือปิดหลอดไฟที่อยู่ตรงแนวทแยงของหลอดไฟที่เสียนั้น ๆ

```
Set broken light (Y/N) ?
Y
Enter row of broken light (0 - 1) =
0
Enter col of broken light (0 - 1) =
0
States in bits = 1111
       | col 0 | col 1
 row 0 |   1x  |   1
 row 1 |   1   |   1

************************************************************
```

2.2 หากป้อน **N** หมายถึง **No** คือไม่ต้องการใส่หลอดไฟที่เสียเข้าไป

```
Set broken light (Y/N) ?
N

************************************************************
```

3. หลังจากนั้นโปรแกรมจะแสดงจำนวนครั้งที่ทำการเปิดปิดไฟได้เร็วที่สุด และจะแสดงการเปิดหรือปิดหลอดไฟแต่ละครั้งพร้อมแสดงกระดานหลังจากการเปิดปิดไฟครั้งนั้นๆ

```
4 moves to turn off all lights

>>> Move 1 : turn off row 1, col 1
States in bits = 1000
       | col 0 | col 1
 row 0 |   1   |   0
 row 1 |   0   |   0

>>> Move 2 : turn on  row 1, col 0
States in bits = 0011
       | col 0 | col 1
 row 0 |   0   |   0
 row 1 |   1   |   1

>>> Move 3 : turn on  row 0, col 1
States in bits = 1110
       | col 0 | col 1
 row 0 |   1   |   1
 row 1 |   1   |   0

>>> Move 4 : turn off row 0, col 0
States in bits = 0000
       | col 0 | col 1
 row 0 |   0   |   0
 row 1 |   0   |   0
```

3.1 หากโปรแกรมไม่มีวิธีแก้ปัญหา โปรแกรมจะแสดงผลออกมาว่า “Cannot solve” หรือโปรแกรมไม่สามารถแก้ปัญหาได้

```
Cannot solve
```

4. โปรแกรมจะถามว่าต้องการเล่นต่อหรือไม่ ซึ่งจะต้องตอบโดยการป้อน **Y** หรือ **N** ผ่าน Keyboard โดยหากตอบ Y ก็คือต้องการเล่นต่อ แต่หากตอบ N จะเป็นการจบโปรแกรม

โครงสร้างโปรแกรม (Programming Structure)

โครงสร้างโปรแกรมและตัวแปรประเภทต่าง ๆ ที่ใช้ในโปรแกรม สามารถอธิบายได้ตามคลาส ดังนี้

1. **Class Light {..}**

```java
public class Light {
    String present = "";
    ArrayList<String> previous = new ArrayList();

    public Light(String p) {...}

    public void setprevious(String pv, int button) { this.previous.set(button, pv); }

    public void setbroken(int po) {...}

    public String getpresent() { return this.present; }

    public String gettoggle(Light pre) {...}

    public int getpo(Light pre) {
        int po = this.previous.indexOf(pre.getpresent());
        return po;
    }

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

    public int hashCode() { return Objects.hash(new Object[]{this.present}); }

    public String toString() { return this.present; }
}
```

โดยสามารถอธิบายเมธอดของโปรแกรมได้ดังนี้

1. public void setprevious(String pv, int button) เป็นเมธอดที่ใช้รับปุ่มก่อนหน้า
2. public void setbroken(int po) เป็นเมธอดที่ใช้รับค่าหลอดไฟที่เสีย
3. public String getpresent() เป็นเมธอดที่ใช้รับค่าสถานะหลอดไฟปัจจุบัน
4. public String gettoggle(Light pre) เป็นเมธอดที่ใช้รับค่าปิด/เปิดหลอดไฟ
5. public int getpo(Light pre) เป็นเมธอดที่ใช้รับค่าตำแหน่งของหลอดไฟ ณ ปัจจุบัน
6. public boolean equals(Object o) เป็นเมธอดที่ถูก Override
7. public int hashCode() เป็นเมธอดที่ที่ถูก Override

2. Class Game {..}

```
public class Game {

    private int N;
    private int allnum;
    private Light initial;
    private Light finish;
    private Graph<Light, DefaultEdge> G;

    public Game(String in, int n) {...}

    public void Start() {...}

    public void addbroken(int row, int col) {...}

    public void showsol(List sol) {...}

    public void printBoard(Light L) {...}

    public String Totoggle(Light L, int i) {...}

    public char toggle(char a) {...}
}
```

โดยสามารถอธิบายเมธอดของโปรแกรมได้ดังนี้

1. public void start() เป็นเมธอดที่ใช้ดำเนินการแก้ปัญหา light out
2. public void addbroken(int row, int col) เป็นเมธอดที่ใช้ในการรับค่าหลอดไฟที่เสีย
3. public void showsol(List sol) เป็นเมธอดที่ใช้แสดงผลขั้นตอนการทำงานของโปรแกรม
4. public void printBoard(Light L) เป็นเมธอดที่ใช้ในการแสดงผลของโปรแกรม
5. public String Totoggle(Light L, int i) เป็นเมธอดที่ใช้หลอดไฟ
6. public char toggle(char a) เป็นเมธอดที่ใช้เปลี่ยนหลอดไฟที่เสีย

3. Class Project2 {..}

```java
public class Project2 {
    public Project2() {
    }

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        boolean repeat = true;
        int size_number = 0;

        label160:
        for(String initial = ""; repeat; System.out.println("\n**********************************************************
            boolean error = true;

            while(error) {
                try {
                    System.out.println("Enter number of rows square grid = ");
                    size_number = Integer.parseInt(scan.nextLine());
                    if (size_number <= 1) {
                        System.out.println("Number of rows square grid must be more than 1");
                    } else {
                        error = false;
                    }
                } catch (InputMismatchException var16) {
                    System.out.println(var16);
                } catch (NumberFormatException var17) {
                    System.out.println(var17);
                }
            }
```

โดยใน Class Project2 {..} มีเพียงเมธอดเดียว นั้นคือ public static void main (String[] args) ซึ่งเป็นเมธอดที่เป็นส่วนหลักในการดำเนินการหรือสั่งทำงานโปรแกรม โดยเมธอดนี้จะวนรับขนาดของตาราง , ถามหา initial states , ตั้งค่า broken light เป็นต้น

รูปแบบ Data structure ที่ใช้งาน

จากโปรแกรมได้มีการเลือกใช้ String ในการทำหน้าที่เก็บข้อมูลหลอดไฟทั้งหมดในกระดานนั้นๆ โดยจะใช้งาน String เปรียบเสมือนกับเป็น Array ของ Char แล้วให้ Char แต่ละตัวแทนหลอดไฟแต่ละดวงและให้ index ของ Char นั้นๆใน String แทนตำแหน่งของหลอดไฟดวงนั้น ๆ ในกระดาน โดยในการเก็บข้อมูลหลอดไฟแต่ละดวงจะเก็บเป็นรูปแบบดังนี้

- หลอดไฟธรรมดาที่ปิดจะเก็บค่าด้วย '0'
- หลอดไฟธรรมดาที่เปิดจะเก็บค่าด้วย '1'
- หลอดไฟเสียที่ปิดจะเก็บค่าด้วย '2'
- หลอดไฟเสียที่เปิดจะเก็บค่าด้วย '3'

หลังจากนั้นก็จะเรียงตามตำแหน่งจากซ้ายไปขวา แถวบนลงล่าง เช่น String เก็บค่า "1120" นั่นหมายความว่ากระดานนี้มีหลอดไฟ 4 ดวงแบบ 2 มิติ ซึ่งในคอลัมน์ที่ 0 แถวที่ 0 มีหลอดไฟธรรมดาที่เปิด คอลัมน์ที่ 1 แถวที่ 0 มีหลอดไฟธรรมดาที่เปิด คอลัมน์ที่ 0 แถวที่ 1 มีหลอดไฟเสียที่ปิด และในคอลัมน์ที่ 1 แถวที่ 1 มีหลอดไฟธรรมดาที่ปิด

และโปรแกรมได้มีการเลือกใช้ Array List ในการเก็บข้อมูลในแต่ละ Initial State จากนั้นข้อมูลจากส่วนนี้จะถูกนำไปสร้างเป็นกราฟแบบ Simple Directed Graph โดยโปรแกรมจะแก้ปัญหา Light out puzzle โดยใช้ Dijkstra Path Algorithm ซึ่งจะมีขั้นตอนดังต่อไปนี้

1. ให้เริ่มต้นจากสถานะเริ่มต้น (Initial State) ของ Light out puzzle
2. ตรวจสอบว่าจุดใดบนตารางหลอดไฟเปิดอยู่
3. สลับสถานะของหลอดไฟที่อยู่ในช่วงเดียวกันแนวนอนหรือแนวตั้งกับหลอดไฟที่เปิดอยู่
4. ทำซ้ำขั้นตอน 2-3 จนกว่าจะไม่มีหลอดไฟที่เปิดอยู่บนตารางหลอดไฟเหลืออยู่
5. ตรวจสอบว่าสถานะปัจจุบันของ Light out puzzle เป็นไปตามเงื่อนไขของการเล่น Light out puzzle หรือไม่
6. ถ้าสถานะปัจจุบันไม่เป็นไปตามเงื่อนไข ให้กลับไปทำซ้ำขั้นตอนที่ 2-4 โดยใช้จุดเริ่มต้นของการแก้ปัญหาก่อนหน้านี้
7. Greedy algorithm เป็น algorithm ที่ใช้เทคนิคการเลือกการกระทำที่ดูเหมือนจะดีที่สุดในขณะนั้น ๆ โดยไม่พิจารณาผลกระทบของการกระทำในขั้นตอนต่อไปโดยละเว้น อย่างไรก็ตาม แม้ว่าอัลกอริทึมนี้จะไม่สามารถแก้ปัญหา Light out puzzle ได้อย่างแม่นยำทั้งหมด แต่ก็มีประสิทธิภาพในระดับหนึ่ง

Demonstrate

```java
public void Start() {
        ShortestPathAlgorithm<Light, DefaultEdge> shortestPath = null;
        shortestPath = new DijkstraShortestPath<>(G);
        List<Light> sol = new ArrayList<Light>();
        ArrayList<Light> AL = new ArrayList<Light>();
        Light temp;
        int i = 0;
        AL.add(initial);
        while (i < AL.size()) {
            temp = AL.get(i);
            if (!G.containsVertex(temp)) {
                G.addVertex(temp);
            }
            for (int j = 0; j < allnum; j++) {
                String newstate = Totoggle(temp, j);
                boolean have = false;
                for (Light vertex : G.vertexSet()) {
                    if (vertex.getpresent().equals(newstate)) {
                        vertex.setprevious(temp.getpresent(), j);
                        G.addEdge(temp, vertex);
                        have = true;
                    }
                }
                if (!have) {
                    Light newlight = new Light(newstate);
                    G.addVertex(newlight);
                    AL.add(newlight);
                    G.addEdge(temp, newlight);
                }
            }
            i++;
        }
        if (!G.containsVertex(finish)) {
            System.out.println("Cannot solve");
        } else {
            sol = shortestPath.getPath(initial, finish).getVertexList();
            showsol(sol);
        }
    }
```

ฟังก์ชันหลักที่ใช้ในการ solve คือ function Start() โดยแนวคิดของโปรแกรมคือการสร้าง graph ที่มี node ที่เก็บทุก ๆ state ของไฟที่เป็นไปได้ทั้งหมดเอาไว้แล้วจากนั้น solve ว่าเราสามารถเปิด/ปิดไฟให้ไฟดับทั้งหมดได้หรือไม่โดยใช้ Shortest path algorithm ของ Djikstra หากใน graph มี node ที่เก็บ state ที่ไฟดับทั้งหมดไว้ก็หมายความว่าเราสามารถปิดไฟทั้งหมดได้นั่นเอง

เมื่อเข้ามาใน function start เราจะ add initial state เก็บไว้ใน ArrayList AL เพื่อเก็บ node เริ่มต้น จากนั้นก็เข้าสู่ while loop ฟังก์ชันนี้จะเช็คไปเรื่อย ๆ ว่าในตอนนี้เก็บทุก state ที่เป็นไปได้หรือยังถ้าไม่มี newstate ถูก add เข้าไป ก็จะหลุดออกจาก while loop นี้

จากนั้นก็จะเข้าสู่ for loop ภายนอกเพื่อสร้าง new state ที่เกิดจากการ toggle ไฟแต่ละดวงให้ครบทุกดวง เมื่อ toggle ไฟดวงนึงแล้วก็จะเข้าสู่ for loop ภายใน เพื่อเช็คว่า graph ในตอนนั้นมี node ที่เก็บ newstate นี้อยู่หรือไม่ ถ้ามีแล้วเราจะ setprevious state ไว้ที่ vertex ที่มี state ของไฟเหมือนกับ newstate และสร้าง edge เชื่อม ระหว่าง temp กับ vertex นั้น ถ้าใน graph ไม่มี newstate ก็จะ add newstate เข้าไปใน ArrayList AL และ สร้าง node ใหม่ที่เก็บ newstate จากนั้นสร้าง edge เชื่อมกันระหว่าง temp กับ newstate

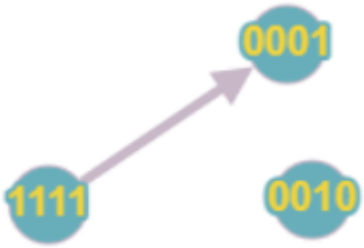
เมื่อผ่านกระบวนการสร้าง Graph ด้านบนแล้วเราจะดูว่า graph นี้มี node ที่เก็บ state ที่ไฟทุกดวงดับไหม(finish) ถ้ามีจะใช้ shortest path ของ dijkstra เพื่อหาเส้นทางระหว่าง initial และ finish ถ้าไม่มีก็จะแสดง cannot solve

Demo1

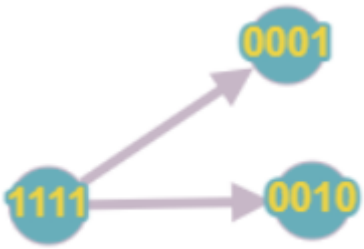
While loop รอบ 1 i = 0
เมื่อเริ่มโปรแกรม initial state คือ 1111 และถูก add เข้าไปใน ArrayList AL
สมาชิก AL = {“1111”}
Temp = Al.get(0) ก็คือ temp = “1111”
ใน while loop เริ่มเช็ค state “1111”
จากนั้นเข้า for loop นอกรอบที่ 1 toggle ไฟแถว 0 คอลัม 0 >>> ได้ new state เป็น 0001

เมื่อเช็ค forloop ในพบว่า newstate ไม่มีอยู่ใน graph จึงเชื่อม temp กับ newstate เข้าด้วยกัน
Add newstate เข้าไปใน ArrayList Al
สมาชิก AL = { “1111” , ”0001” }

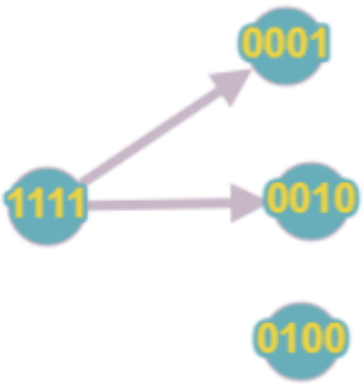
for loop นอกรอบที่ 2 toggle ไฟแถว 0 คอลัม 1 >>> ได้ new state เป็น 0010



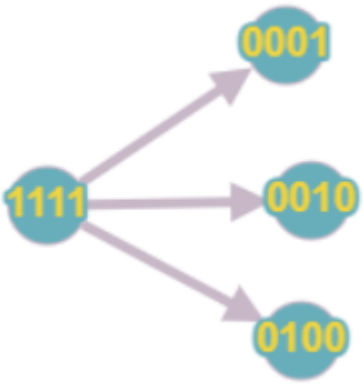
เมื่อเช็ค forloop ใน พบว่า newstate ไม่มีอยู่ใน graph จึงเชื่อม temp กับ newstate เข้าด้วยกัน
Add newstate เข้าไปใน ArrayList Al
สมาชิก AL = { “1111” , ”0001” , “0010” }



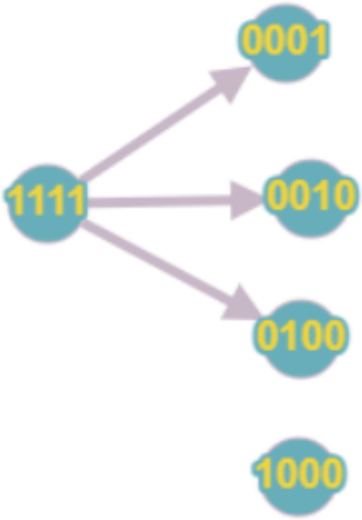
for loop นอกรอบที่ 3 toggle ไฟแถว 1 คอลัม 0 >>> ได้ new state เป็น 0100



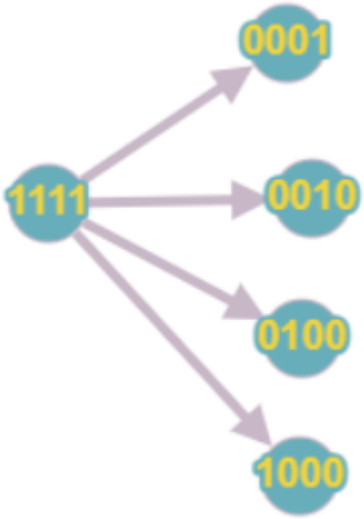
เมื่อเช็ค forloop ในพบว่า newstate ไม่มีอยู่ใน graph จึงเชื่อม temp กับ newstate เข้าด้วยกัน
Add newstate เข้าไปใน ArrayList Al
สมาชิก AL = { “1111” , ”0001” , “0010” , “0100” }



for loop นอกรอบที่ 4 toggle ไฟแถว 1 คอลัม 1 >>> ได้ new state เป็น 1000



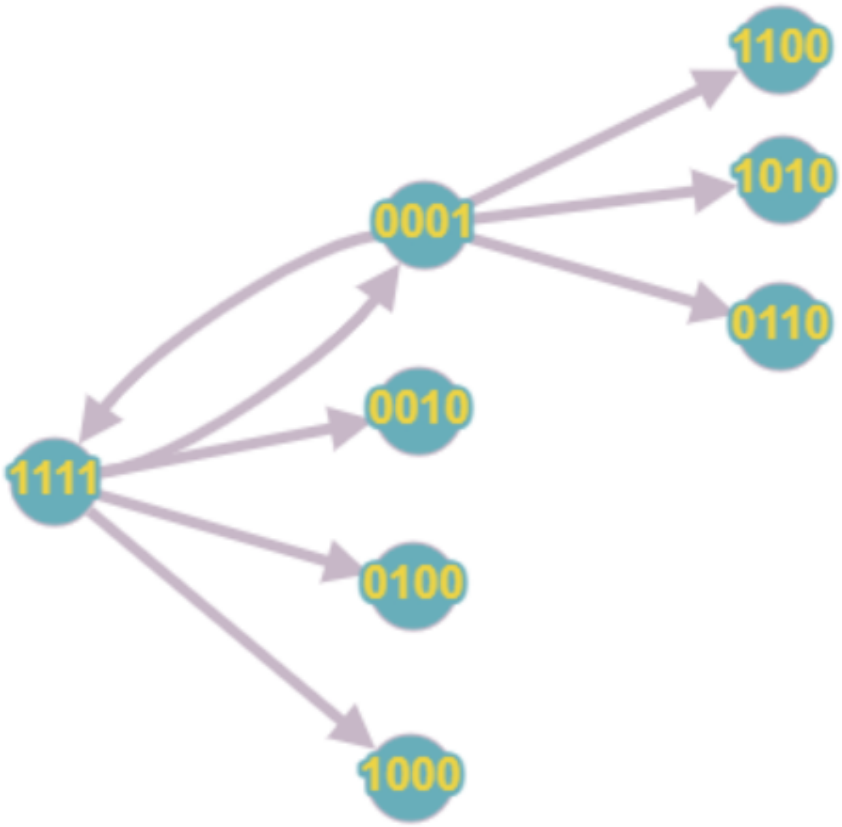
เมื่อเช็ค forloop ในพบว่า newstate ไม่มีอยู่ใน graph จึงเชื่อม temp กับ newstate เข้าด้วยกัน
Add newstate เข้าไปใน ArrayList Al
สมาชิก AL = { “1111” , ”0001” , “0010” , “0100” , “1000” }



While loop รอบที่ 2 i = 1

Temp = Al.get(1) ดังนั้น temp = “0001”

ทำแบบเดียวกับการกระบวนการข้างต้น เมื่อ toggle ไฟทั้ง 4 ดวง จะได้ state “1111”,”1100”,”1010”,”0110”



สมาชิก AL = { “1111” , ”0001” , “0010” , “0100” , “1000” , “1100” , “1010” , “0110” }

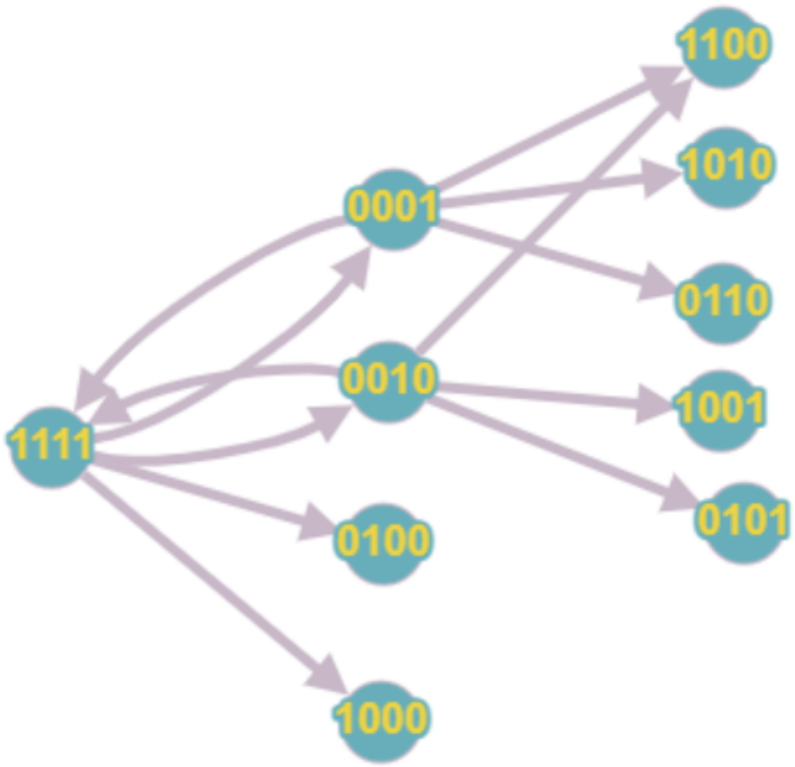
สีฟ้าแทน state ปัจจุบันที่จะ toggle ต่อไป

สีส้มแทน state ใหม่ที่เกิดขึ้นจากการ toggle state ปัจจุบัน ที่ไม่ซ้ำกับ graph

While loop รอบที่ 3 i = 2

Temp = Al.get(2) ดังนั้น temp = “0010”

ทำแบบเดียวกับการกระบวนการข้างต้น เมื่อ toggle ไฟทั้ง 4 ดวง จะได้ state “1111”,”1100”,”1001”,”0101”



สมาชิก AL = { “1111” , ”0001” , “0010” , “0100” , “1000” , “1100” , “1010” ,
“0110”,”1001”,”0101” }

สีฟ้าแทน state ปัจจุบันที่จะ toggle ต่อไป

สีส้มแทน state ใหม่ที่เกิดขึ้นจากการ toggle state ปัจจุบัน ที่ไม่ซ้ำกับ graph

While loop รอบที่ 4 i = 3

Temp = Al.get(2) ดังนั้น temp = “0100”

ทำแบบเดียวกับการกระบวนการข้างต้น เมื่อ toggle ไฟทั้ง 4 ดวง จะได้ state “1111”,”1010”,”1001”,”0011”

สมาชิก AL = { “1111” , ”0001” , “0010” , “0100” , “1000” , “1100” , “1010” ,
“0110”,”1001”,”0101”,”0011” }

สีฟ้าแทน state ปัจจุบันที่จะ toggle ต่อไป

สีส้มแทน state ใหม่ที่เกิดขึ้นจากการ toggle state ปัจจุบัน ที่ไม่ซ้ำกับ graph

While loop รอบที่ 5 i = 4

Temp = Al.get(4) ดังนั้น temp = “1000”

ทำแบบเดียวกับการกระบวนการข้างต้น เมื่อ toggle ไฟทั้ง 4 ดวง จะได้ state “1111”,”0110”,”0101”,”0011”

สมาชิก AL = { “1111” , ”0001” , “0010” , “0100” , “1000” , “1100” , “1010” ,
“0110”,”1001”,”0101”,”0011” }

สีฟ้าแทน state ปัจจุบันที่จะ toggle ต่อไป

สีส้มแทน state ใหม่ที่เกิดขึ้นจากการ toggle state ปัจจุบัน ที่ไม่ซ้ำกับ graph

While loop รอบที่ 6 i = 5

Temp = Al.get(5) ดังนั้น temp = “1100”

ทำแบบเดียวกับการกระบวนการข้างต้น เมื่อ toggle ไฟทั้ง 4 ดวง จะได้ state

“0010”,”0001”,”1011”,”0111”

หลังจากนี้ขอใช้ simple graph แทน edge ขาไปและขากลับเพื่อไม่ให้ edge วกไปวนมา และป้องกันความสับสน

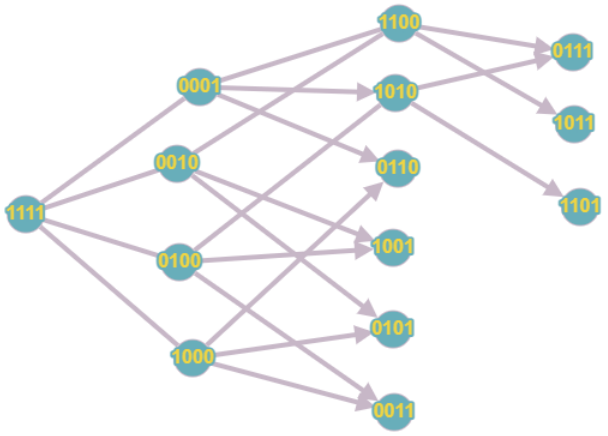
สมาชิก AL = { “1111” , ”0001” , “0010” , “0100” , “1000” , “1100” , “1010” ,
“0110”,”1001”,”0101”,”0011”,”0111”,”1011” }

While loop รอบที่ 7 i = 6

Temp = Al.get(6) ดังนั้น temp = “1010”

ทำแบบเดียวกับการกระบวนการข้างต้น เมื่อ toggle ไฟทั้ง 4 ดวง จะได้ state

“0001”,”0100”,”0111”,”1101”

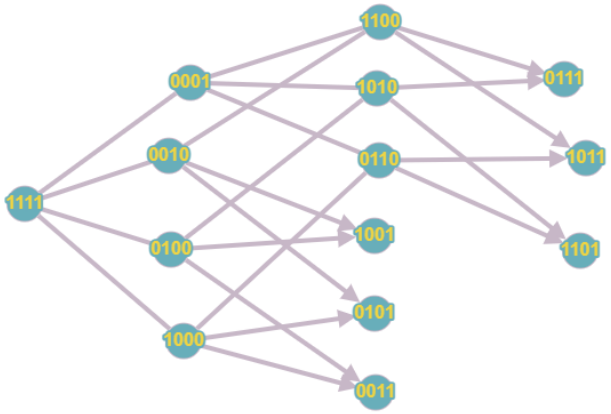


สมาชิก AL = { “1111” , ”0001” , “0010” , “0100” , “1000” , “1100” , “1010” ,
“0110”,”1001”,”0101”,”0011”,”0111”,”1011””,”1101” }

While loop รอบที่ 8 i = 7

Temp = Al.get(7) ดังนั้น temp = “0110”

ทำแบบเดียวกับการกระบวนการข้างต้น เมื่อ toggle ไฟทั้ง 4 ดวง จะได้ state “0001”,”1000”,”1101”,”1011”

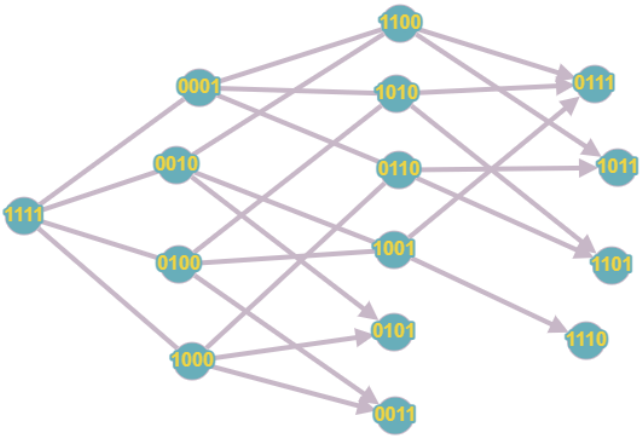


สมาชิก AL = { “1111” , ”0001” , “0010” , “0100” , “1000” , “1100” , “1010” ,
“0110”,”1001”,”0101”,”0011”,”0111”,”1011””,”1101” }

While loop รอบที่ 9 i = 8

Temp = Al.get(8) ดังนั้น temp = “1001”

ทำแบบเดียวกับการกระบวนการข้างต้น เมื่อ toggle ไฟทั้ง 4 ดวง จะได้ state “0010”,”0100”,”0111”,”1110”

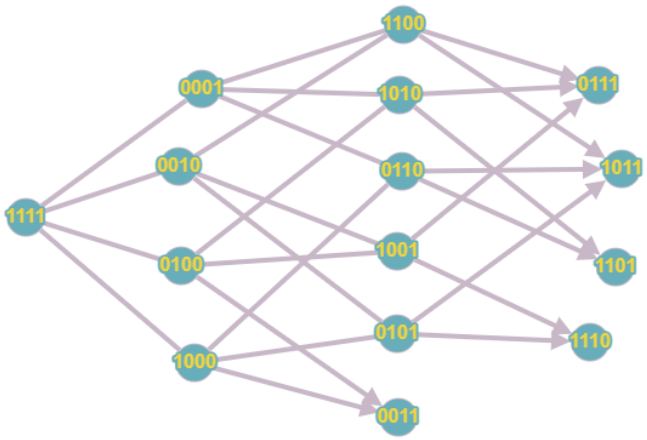


สมาชิก AL = { “1111” , ”0001” , “0010” , “0100” , “1000” , “1100” , “1010” ,
“0110”,”1001”,”0101”,”0011”,”0111”,”1011””,”1101”,”1110” }

While loop รอบที่ 10 i = 9

Temp = Al.get(9) ดังนั้น temp = “0101”

ทำแบบเดียวกับการกระบวนการข้างต้น เมื่อ toggle ไฟทั้ง 4 ดวง จะได้ state
“0010”,”1000”,”1011”,”1110”

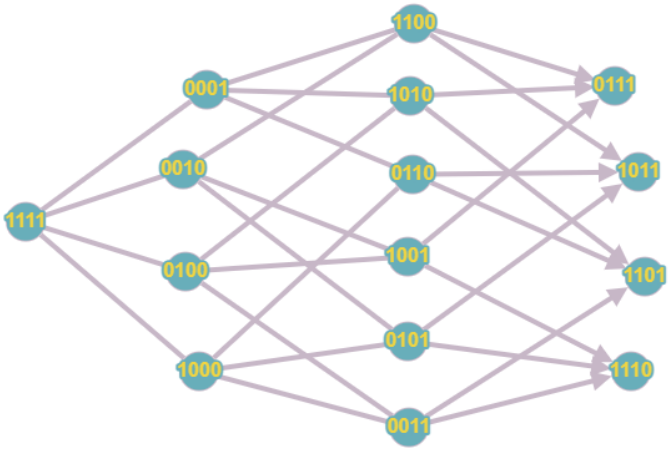


สมาชิก AL = { “1111” , ”0001” , “0010” , “0100” , “1000” , “1100” , “1010” ,
“0110”,”1001”,”0101”,”0011”,”0111”,”1011””,”1101”,”1110” }

While loop รอบที่ 11 i = 10

Temp = Al.get(10) ดังนั้น temp = “0011”

ทำแบบเดียวกับการกระบวนการข้างต้น เมื่อ toggle ไฟทั้ง 4 ดวง จะได้ state “0100”,”1000”,”1101”,”1110”

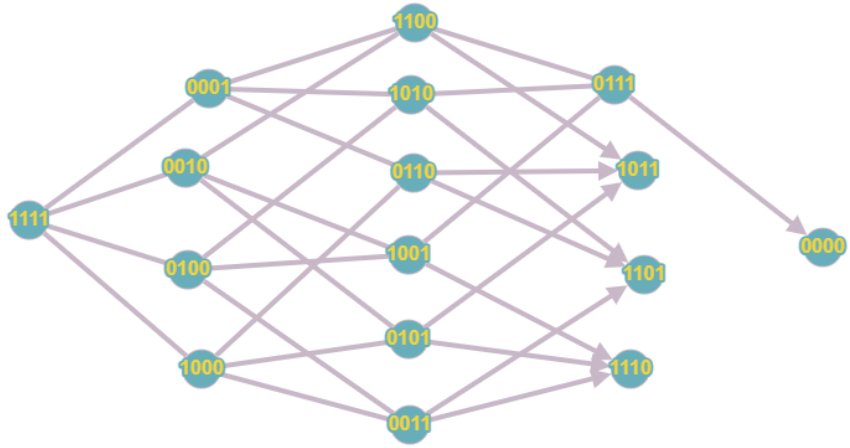


สมาชิก AL = { “1111” , ”0001” , “0010” , “0100” , “1000” , “1100” , “1010” ,
“0110”,”1001”,”0101”,”0011”,”0111”,”1011””,”1101”,”1110” }

While loop รอบที่ 12 i = 11

Temp = Al.get(11) ดังนั้น temp = “0111”

ทำแบบเดียวกับการกระบวนการข้างต้น เมื่อ toggle ไฟทั้ง 4 ดวง จะได้ state “1100”,”1010”,”1001”,”0000”

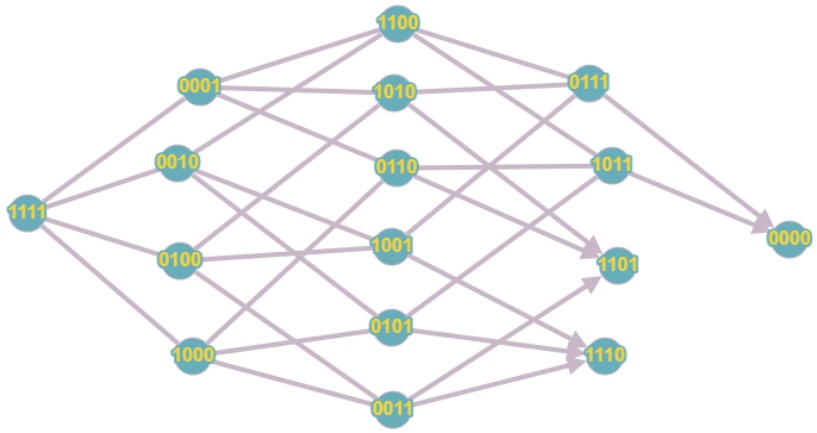


สมาชิก AL = { “1111” , ”0001” , “0010” , “0100” , “1000” , “1100” , “1010” ,
“0110”,”1001”,”0101”,”0011”,”0111”,”1011””,”1101”,”1110”,”0000” }

While loop รอบที่ 13 i = 12

Temp = Al.get(12) ดังนั้น temp = “1011”

ทำแบบเดียวกับการกระบวนการข้างต้น เมื่อ toggle ไฟทั้ง 4 ดวง จะได้ state “1100”,”0110”,”0101”,”0000”

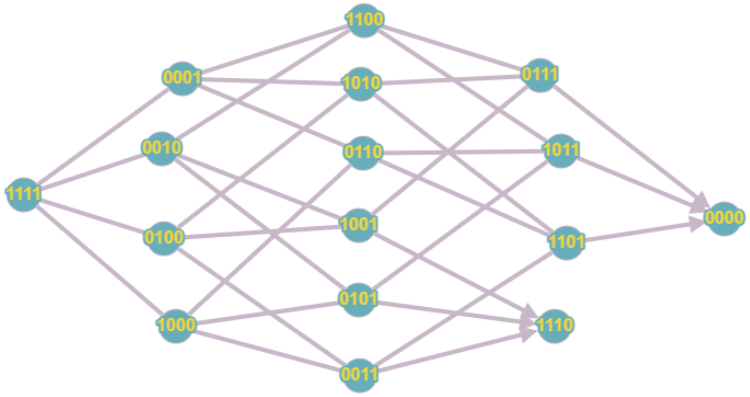


สมาชิก AL = { “1111” , ”0001” , “0010” , “0100” , “1000” , “1100” , “1010” ,
“0110”,”1001”,”0101”,”0011”,”0111”,”1011””,”1101”,”1110”,”0000” }

While loop รอบที่ 14 i = 13

Temp = Al.get(13) ดังนั้น temp = “1101”

ทำแบบเดียวกับการกระบวนการข้างต้น เมื่อ toggle ไฟทั้ง 4 ดวง จะได้ state “1010”,”0110”,”0011”,”0000”

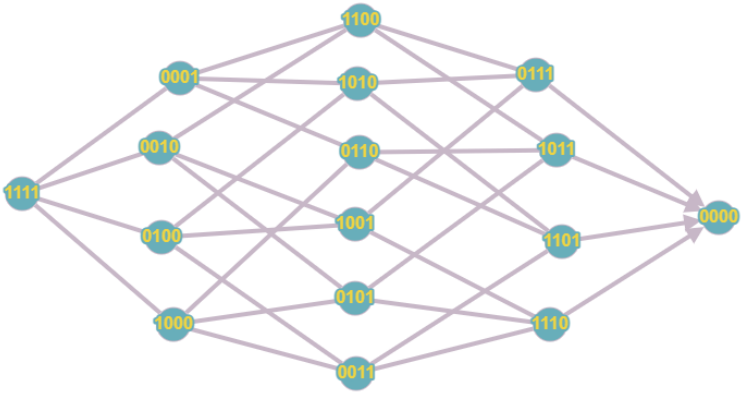


สมาชิก AL = { “1111” , ”0001” , “0010” , “0100” , “1000” , “1100” , “1010” ,
“0110”,”1001”,”0101”,”0011”,”0111”,”1011””,”1101”,”1110”,”0000” }

While loop รอบที่ 15 i = 14

Temp = Al.get(12) ดังนั้น temp = “1110”

ทำแบบเดียวกับการกระบวนการข้างต้น เมื่อ toggle ไฟทั้ง 4 ดวง จะได้ state “1001”,”0101”,”0011”,”0000”

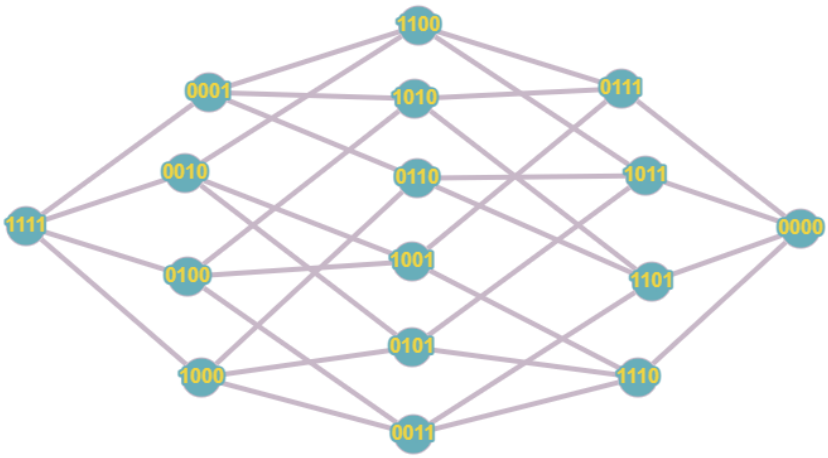


สมาชิก AL = { “1111” , ”0001” , “0010” , “0100” , “1000” , “1100” , “1010” , “0110”,”1001”,”0101”,”0011”,”0111”,”1011””,”1101”,”1110”,”0000” }

While loop รอบที่ 16 i = 15

Temp = Al.get(15) ดังนั้น temp = “0000”

ทำแบบเดียวกับการกระบวนการข้างต้น เมื่อ toggle ไฟทั้ง 4 ดวง จะได้ state “0111”,”1011”,”1101”,”1110”



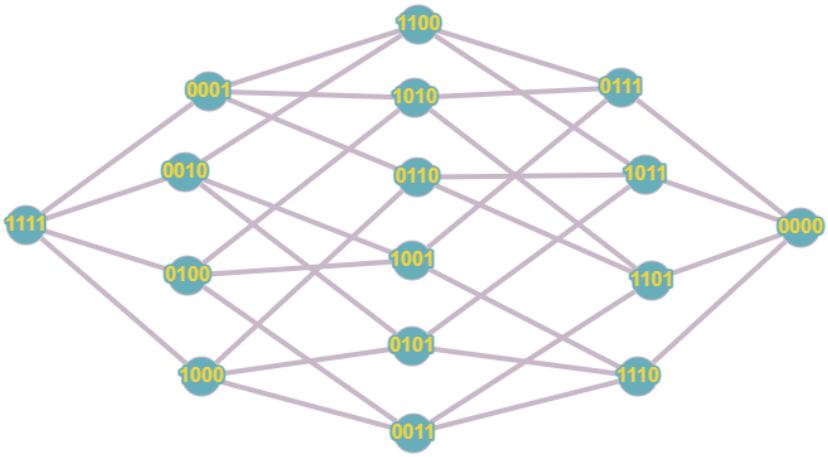
สมาชิก AL = { “1111” , ”0001” , “0010” , “0100” , “1000” , “1100” , “1010” , “0110”,”1001”,”0101”,”0011”,”0111”,”1011””,”1101”,”1110”,”0000” }

While loop รอบที่ 17 i = 16

จะพบว่า เงื่อนไข i< Al.size() เป็น false จึงหลุดออกจาก while loop แล้วรันไปเรื่อย ๆ จนถึง code ด้านล่าง

```
        }
        if (!G.containsVertex(finish)) {
            System.out.println("Cannot solve");
        } else {
            sol = shortestPath.getPath(initial, finish).getVertexList();
            showsol(sol);
        }
    }
```

Graph สุดท้ายที่ได้จะเป็นรูปแบบนี้โดย undirect graph แทนรูปแบบที่ว่า state แต่ละ state สามารถเปิดปิดไฟเพื่อไปอีก state ได้ทั้งขาไปและขากลับ สาเหตุที่ทำแบบนี้เพื่อลดความสับสนจากการดู edge ที่วกไปวนมา แต่ในตัวโปรแกรมจริงจะเป็น direct graph



Graph Demo1

จาก graph จะเห็นว่ามี node ที่มี state ที่ไฟดับทั้งหมด (“0000”) อยู่ดังนั้นเราจึงสามารถหา shortest path ได้ โดยเงื่อนไขที่ได้จาก G.containvertex(finish) จะ return true ดังนั้น !G.containvertex(finish) เป็น false โปรแกรมจะไม่ทำงานใน scope if แต่ทำงานใน else แล้วเริ่มหา shortest path

```
>>> Move 1 : turn on  row 1, col 1
States in bits = 1000
       | col 0 | col 1
 row 0 |   1   |   0
 row 1 |   0   |   0

>>> Move 2 : turn off row 1, col 0
States in bits = 0011
       | col 0 | col 1
 row 0 |   0   |   0
 row 1 |   1   |   1

>>> Move 3 : turn off row 0, col 1
States in bits = 1110
       | col 0 | col 1
 row 0 |   1   |   1
 row 1 |   1   |   0

>>> Move 4 : turn on  row 0, col 0
States in bits = 0000
       | col 0 | col 1
 row 0 |   0   |   0
 row 1 |   0   |   0
```

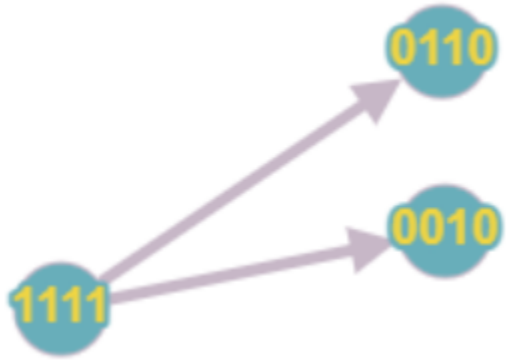
Demo2

While loop รอบ 1 i = 0
เมื่อเริ่มโปรแกรม initial state คือ 1111 และถูก add เข้าไปใน ArrayList AL
สมาชิก AL = {“1111”}
Temp = Al.get(0) ก็คือ temp = “1111”
ใน while loop เริ่มเช็ค state “1111”
จากนั้นเข้า for loop นอกรอบที่ 1 toggle ไฟแถว 0 คอลัม 0 >>> ได้ new state เป็น 0110

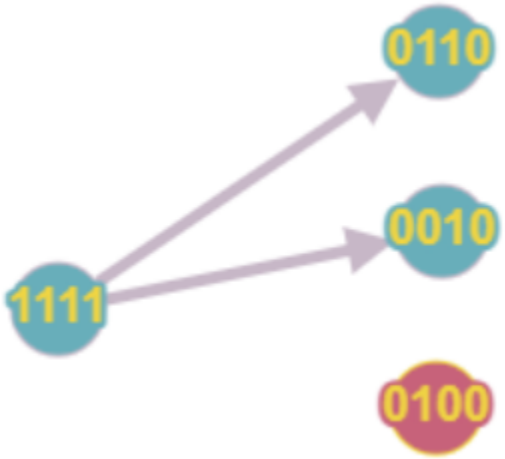
เมื่อเช็ค forloop ในพบว่า newstate ไม่มีอยู่ใน graph จึงเชื่อม temp กับ newstate เข้าด้วยกัน
Add newstate เข้าไปใน ArrayList Al
สมาชิก AL = { “1111” , ”0110” }

for loop นอกรอบที่ 2 toggle ไฟแถว 0 คอลัม 1 >>> ได้ new state เป็น 0010

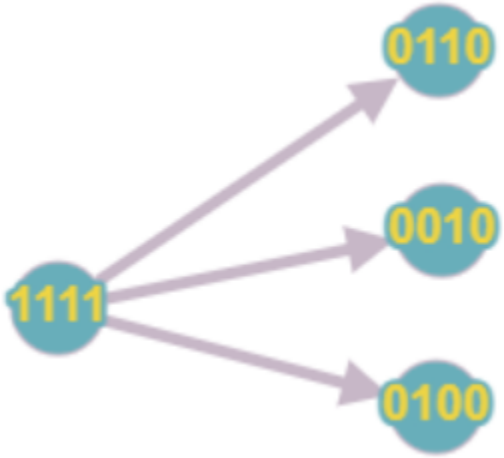
เมื่อเช็ค forloop ในพบว่า newstate ไม่มีอยู่ใน graph จึงเชื่อม temp กับ newstate เข้าด้วยกัน
Add newstate เข้าไปใน ArrayList Al
สมาชิก AL = { “1111” , ”0110”,”0010” }



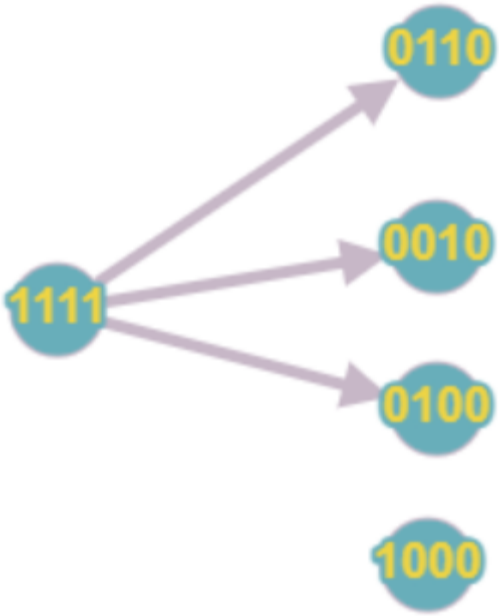
for loop นอกรอบที่ 3 toggle ไฟแถว 1 คอลัม 0 >>> ได้ new state เป็น 0100



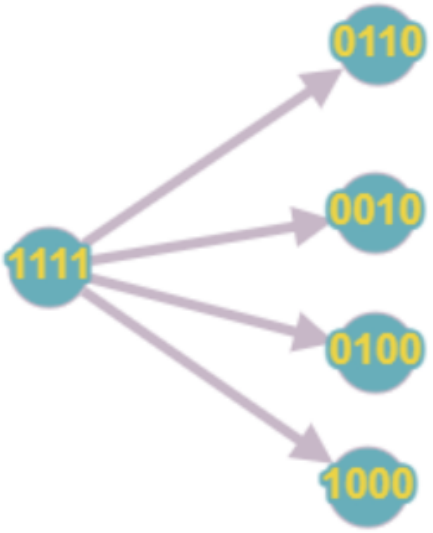
เมื่อเช็ค forloop ในพบว่า newstate ไม่มีอยู่ใน graph จึงเชื่อม temp กับ newstate เข้าด้วยกัน
Add newstate เข้าไปใน ArrayList Al
สมาชิก AL = { “1111” , ”0110”,”0010”,”0100” }



for loop นอกรอบที่ 4 toggle ไฟแถว 1 คอลัม 1 >>> ได้ new state เป็น 1000



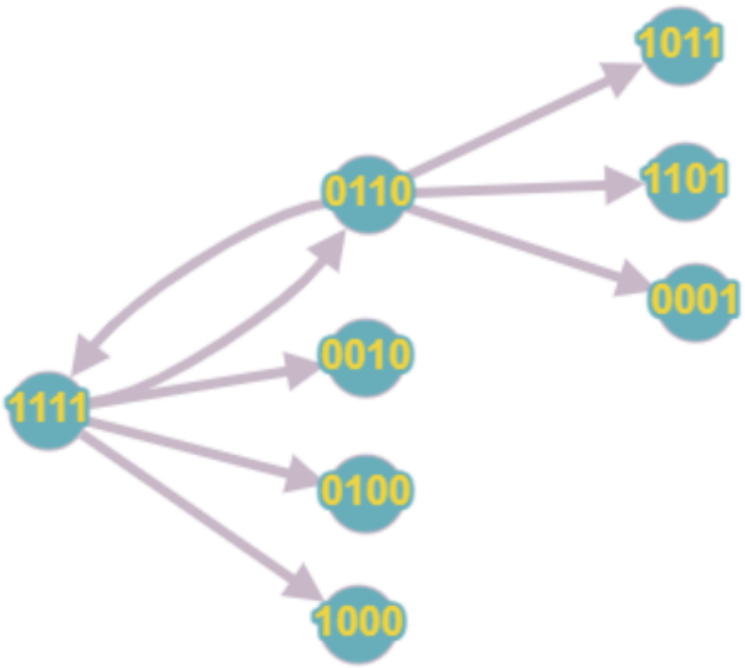
เมื่อเช็ค forloop ในพบว่า newstate ไม่มีอยู่ใน graph จึงเชื่อม temp กับ newstate เข้าด้วยกัน
Add newstate เข้าไปใน ArrayList Al
สมาชิก AL = { “1111” , ”0110”,”0010”,”0100”,”1000” }



While loop รอบที่ 2 i = 1

Temp = Al.get(1) ดังนั้น temp = “0001”

ทำแบบเดียวกับการกระบวนการข้างต้น เมื่อ toggle ไฟทั้ง 4 ดวง จะได้ state “1111”,”1011”,”1101”,”0001”

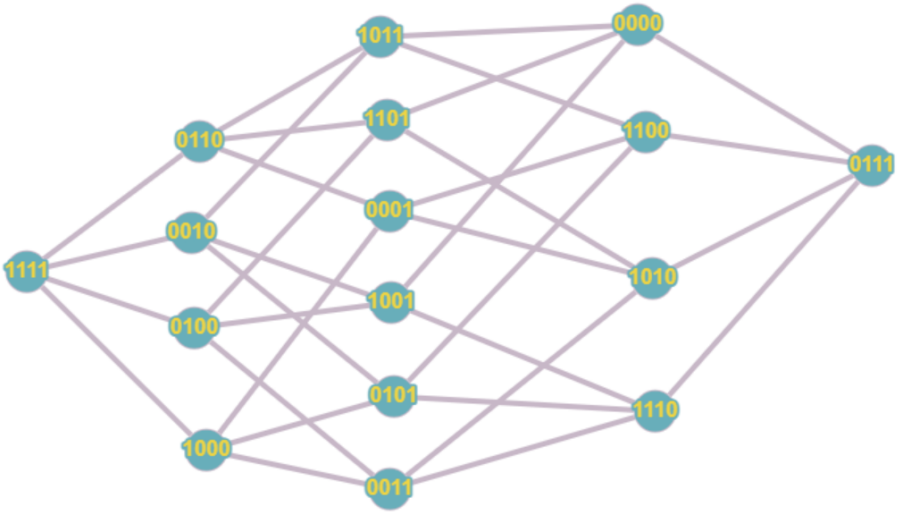


สมาชิก AL = { “1111” , ”0110”,”0010”,”0100”,”1000”,”1011”,”1101”,”0001” }

สีฟ้าแทน state ปัจจุบันที่จะ toggle ต่อไป

สีส้มแทน state ใหม่ที่เกิดขึ้นจากการ toggle state ปัจจุบัน ที่ไม่ซ้ำกับ graph

จากนั้นก็ทำแบบนี้ไปเรื่อย ๆ แบบเดียวกับ Demo1 จนกว่าจะหลุดออกจาก while loop และgraph สุดท้ายที่ได้จะเป็นรูปแบบนี้โดย undirect graph แทนรูปแบบที่ว่า state แต่ละ state สามารถเปิดปิดไฟเพื่อไปอีก state ได้ทั้งขาไปและขากลับ สาเหตุที่ทำแบบนี้เพื่อลดความสับสนจากการดู edge ที่วกไปวนมาและโปรแกรมแต่ละ node จะเก็บตัวแปรไว้มากกว่านี้แต่เพื่อความเข้าใจง่ายจึงใช้ตัวแทนของแต่ละ node เป็น state ของไฟ



Graph Demo2

จะเห็นว่ามี node ที่มี state ที่ไฟดับทั้งหมด (“0000”) อยู่ ดังนั้น เราจึงสามารถหา shortest path ได้

```
>>> Move 1 : turn on  row 0, col 0
States in bits = 0110
       | col 0 | col 1
 row 0 |   0x  |   1
 row 1 |   1   |   0

>>> Move 2 : turn on  row 1, col 0
States in bits = 1101
       | col 0 | col 1
 row 0 |   1x  |   1
 row 1 |   0   |   1

>>> Move 3 : turn on  row 0, col 1
States in bits = 0000
       | col 0 | col 1
 row 0 |   0x  |   0
 row 1 |   0   |   0
```

Graph Algorithm

เราเลือกใช้ Djikstra Algorithm ในการหาเส้นทางที่สั้นที่สุดจาก state ไฟตั้งต้น จนถึง state ที่ไฟดับทั้งหมด Dijkstra Algorithm นั้นสามารถใช้ได้กับทั้ง weighted และ unweighted graph การทำงานของ Dijkstra ใน unweighted graph จะทำงานๆคล้ายกับ Breath first search **เหตุผลที่ใช้คือ ในอนาคตหากเราต้องการเอา code ปัจจุบันไป implement หรือต่อยอด ไปเป็น weighted graph code ในปัจจุบันก็ยังสามารถทำงานได้ในกรณีที่ไม่มี negative cycle**

เปรียบเทียบ Demo 3,4,5

```
>>> Move 1 : turn off row 0, col 1
States in bits = 111111010
       | col 0 | col 1 | col 2
 row 0 |   1   |   1   |   1
 row 1 |   1   |   1   |   1
 row 2 |   0   |   1   |   0x

>>> Move 2 : turn on  row 2, col 1
States in bits = 111101101
       | col 0 | col 1 | col 2
 row 0 |   1   |   1   |   1
 row 1 |   1   |   0   |   1
 row 2 |   1   |   0   |   1x

>>> Move 3 : turn on  row 0, col 2
States in bits = 100100101
       | col 0 | col 1 | col 2
 row 0 |   1   |   0   |   0
 row 1 |   1   |   0   |   0
 row 2 |   1   |   0   |   1x

>>> Move 4 : turn on  row 2, col 2
States in bits = 100110100
       | col 0 | col 1 | col 2
 row 0 |   1   |   0   |   0
 row 1 |   1   |   1   |   0
 row 2 |   1   |   0   |   0x

>>> Move 5 : turn on  row 1, col 0
States in bits = 000000000
       | col 0 | col 1 | col 2
 row 0 |   0   |   0   |   0
 row 1 |   0   |   0   |   0
 row 2 |   0   |   0   |   0x
```

Demo3 : no breaklight

```
>>> Move 1 : turn on  row 1, col 0
States in bits = 100011110
       | col 0 | col 1 | col 2
 row 0 |   1   |   0   |   0
 row 1 |   0   |   1   |   1
 row 2 |   1   |   1   |   0

>>> Move 2 : turn on  row 1, col 2
States in bits = 101000111
       | col 0 | col 1 | col 2
 row 0 |   1   |   0   |   1
 row 1 |   0   |   0   |   0
 row 2 |   1   |   1   |   1

>>> Move 3 : turn off row 0, col 1
States in bits = 010010111
       | col 0 | col 1 | col 2
 row 0 |   0   |   1   |   0
 row 1 |   0   |   1   |   0
 row 2 |   1   |   1   |   1

>>> Move 4 : turn on  row 2, col 2
States in bits = 010011100
       | col 0 | col 1 | col 2
 row 0 |   0   |   1   |   0
 row 1 |   0   |   1   |   1
 row 2 |   1   |   0   |   0

>>> Move 5 : turn on  row 1, col 1
States in bits = 000100110
       | col 0 | col 1 | col 2
 row 0 |   0   |   0   |   0
 row 1 |   1   |   0   |   0
 row 2 |   1   |   1   |   0

>>> Move 6 : turn on  row 2, col 0
States in bits = 000000000
       | col 0 | col 1 | col 2
 row 0 |   0   |   0   |   0
 row 1 |   0   |   0   |   0
```

Demo4 : breaklight

```
States in bits = 000101010
       | col 0 | col 1 | col 2
 row 0 |   0   |   0   |   0
 row 1 |   1   |   0   |   1
 row 2 |   0   |   1   |   0

************************************************************

Set broken light (Y/N) ?
y
Enter row of broken light (0 - 2) =
0
Enter col of broken light (0 - 2) =
1
States in bits = 000101010
       | col 0 | col 1 | col 2
 row 0 |   0   |   0x  |   0
 row 1 |   1   |   0   |   1
 row 2 |   0   |   1   |   0

************************************************************
Cannot solve
```

Demo5 : breaklight

หลังจากในแต่ละ demo ได้สร้าง graph เสร็จเรียบร้อยแล้ว ทางคณะผู้จัดทำจึงสร้าง function ข้างล่างเพื่อเปรียบเทียบความแตกต่างระหว่าง demo3,4,5

```java
int round = 1;
        for (Light vertex : G.vertexSet()) {

            System.out.printf("   %s   ", vertex.getpresentconverse());
            if (round == 7) {
                System.out.println("");
                round = 0;
            }

            round++;
        }
```

Function นี้มีเพื่อแสดงค่า state ที่เก็บไว้ในทุก ๆ node เพื่อนำ outputมาเปรียบเทียบกัน

- All state from demo3

```
************************************************************
 000101010  110001010  111111010  011100010  100011110  010010000  001110011
 000001100  000111101  000100001  001011010  101000010  010111110  100110000
 111010011  110101100  110011101  110000001  100110010  011001110  101000000
 110100011  111011100  111101101  111110001  111010110  001011000  010111011
 011000100  011110101  011101001  110100100  101000111  100111000  100001001
 100010101  011001001  010110110  010000111  010011011  001010101  001100100
 001111000  000011011  000000111  000110110  010010010  101101110  011100000
 000000011  001111100  001001101  001010001  001110110  111111000  100011011
 101100100  101010101  101001001  000000100  011100111  010011000  010101001
 010110101  101101001  100010110  100100111  100111011  111110101  111000100
 111011000  110111011  110100111  110010110  000000110  110001000  101101011
 100010100  100100101  100111001  001110100  010010111  011101000  011011001
 011000101  100011001  101100110  101010111  101001011  110000101  110110100
 110101000  111001011  111010111  111100110  101101100  110001111  111110000
 111000001  111011101  000000001  001111110  001001111  001010011  010011101
 010101100  010110000  011010011  011001111  011111110  111111101  110000010
 110110011  110101111  101100001  101010000  101001100  100101111  100110011
 100000010  011101111  011011110  011000010  010100001  010111101  010001100
 001000010  001011110  001101111  000010000  110100110  000101000  011001011
 010110100  010000101  010011001  111010100  100110111  101001000  101111001
 101100101  010111001  011000110  011110111  011101011  000100101  000010100
 000001000  001101011  001110111  001000110  011001100  000101111  001010000
 001100001  001111101  110100001  111011110  111101111  111110011  100111101
 100001100  100010000  101110011  101101111  101011110  001011101  000100010
 000010011  000001111  011000001  011110000  011101100  010001111  010010011
 010100010  101001111  101111110  101100010  100000001  100011101  100101100
 111100010  111111110  111001111  110110000  010111100  001011111  000100000
 000010001  000001101  111010001  110101110  110011111  110000011  101001101
 101111100  101100000  100000011  100011111  100101110  000101101  001010010
 001100011  001111111  010110001  010000000  010011100  011111111  011100011
 011010010  100111111  100001110  100010010  101110001  101101101  101011100
 110010010  110001110  110111111  111000000  100110101  101001010  101111011
 101100111  110101001  110011000  110000100  111100111  111111011  111001010
 000100111  000010110  000001010  001101001  001110101  001000100  010001010
 010010110  010100111  011011000  111011011  111101010  111110110  110010101
 110001001  110111000  101110110  101101010  101011011  100100100  011111000
 011100100  011010101  010101010  001001001  100011100  111111111  110000000
```

```
 110110001  110101101  001110001  000001110  000111111  000100011  011101101
 011011100  011000000  010100011  010111111  010001110  110001101  111110010
 111000011  111011111  100010001  100100000  100111100  101011111  101000011
 101110010  010011111  010101110  010110010  011010001  011001101  011111100
 000110010  000101110  000011111  001100000  010010101  011101010  011011011
 011000111  000001001  000111000  000100100  001000111  001011011  001101010
 110000111  110110110  110101010  111001001  111010101  111100100  100101010
 100110110  100000111  101111000  001111011  001001010  001010110  000110101
 000101001  000011000  011010110  011001010  011111011  010000100  101011000
 101000100  101110101  100001010  111101001  011100101  010011010  010101011
 010110111  001111001  001001000  001010100  000110111  000101011  000011010
 111110111  111000110  111011010  110111001  110100101  110010100  101011010
 101000110  101110111  100001000  000001011  000111010  000100110  001000101
 001011001  001101000  010100110  010111010  010001011  011110100  100101000
 100110100  100000101  101111010  110011001  100010011  100100010  100111110
 101011101  101000001  101110000  110111110  110100010  110010011  111101100
 000110000  000101100  000011101  001100010  010000001  111001100  111010000
 111100001  110011110  101111101  011110011  101000101  100111010  100001011
 100010111  111011001  111101000  111110100  110010111  110001011  110111010
 001010111  001100110  001111010  000011001  000000101  000110100  011111010
 011100110  011010111  010101000  110101011  110011010  110000110  111100101
 111111001  111001000  100000110  100011010  100101011  101010100  010001000
 010010100  010100101  011011010  000111001  010110011  010000010  010011110
 011111101  011100001  011010000  000011110  000000010  000110011  001001100
 110010000  110001100  110111101  111000010  100100001  001101100  001110000
 001000001  000111110  011011101  101010011  011000011  011110010  011101110
 010001101  010010001  010100000  001101110  001110010  001000011  000111100
 111100000  111111100  111001101  110110010  101010001  000011100  000000000
 000110001  001001110  010101101  100100011  100000100  100011000  100101001
 101010110  110110101  000111011  111000111  101100011  101010010  101001110
 100101101  100110001  100000000  111001110  111010010  111100011  110011100
 001000000  001011100  001101101  000010010  011110001  110111100  110100000
 110010001  111101110  100001101  010000011  010100100  010111000  010001001
 011110110  000010101  110011011  001100111  011010100  011001000  011111001
 010000110  001100101  111101011  000010111  100001111  101110100  101101000
 101011001  100100110  111000101  001001011  110110111  010101111  011011111
 101111111
```

- All state from demo4

```
000101010 110001010 111111010 011100010 100011110 010010000 001110011
000001100 000111101 000111011 001011010 101000010 010111110 100110000
111010011 110101100 110011101 110011011 100110010 011001110 101000000
110100011 111011100 111101101 111101011 111010110 001011000 010111011
011000100 011110101 011110011 110100100 101000111 100111000 100001001
100001111 011001001 010110110 010000111 010000001 001010101 001100100
001100010 000011011 000011101 000101100 010010010 101101110 011100000
000000011 001111100 001001101 001001011 001110110 111111000 100011011
101100100 101010101 101010011 000000100 011100111 010011000 010101001
010101111 101101001 100010110 100100111 100100001 111110101 111000100
111000010 110111011 110111101 110001100 000000110 110001000 101101011
100010100 100100101 100100011 001110100 010010111 011101000 011011001
011011111 100011001 101100110 101010111 101010001 110000101 110110100
110110010 111001011 111001101 111111100 101101100 110001111 111110000
111000001 111000111 000000001 001111110 001001111 001001001 010011101
010101100 010101010 011010011 011010101 011100100 111111101 110000010
110110011 110110101 101100001 101010000 101010110 100101111 100101001
100011000 011101111 011011110 011011000 010100001 010100111 010010110
001000010 001000100 001110101 000001010 110100110 000101000 011001011
010110100 010000101 010000011 111010100 100110111 101001000 101111001
101111111 010111001 011000110 011110111 011110001 000100101 000010100
000010010 001101011 001101101 001011100 011001100 000101111 001010000
001100001 001100111 110100001 111011110 111101111 111101001 100111101
100001100 100001010 101110011 101110101 101000100 001011101 000100010
000010011 000010101 011000001 011110000 011110110 010001111 010001001
010111000 101001111 101111110 101111000 100000001 100000111 100110110
111100010 111100100 111010101 110101010 010111100 001011111 000100000
000010001 000010111 111010001 110101110 110011111 110011001 101001101
101111100 101111010 100000011 100000101 100110100 000101101 001010010
001100011 001100101 010110001 010000000 010000110 011111111 011111001
011001000 100111111 100001110 100001000 101110001 101110111 101000110
110010010 110010100 110100101 111011010 100110101 101001010 101111011
101111101 110101001 110011000 110011110 111100111 111100001 111010000
000100111 000010110 000010000 001101001 001101111 001011110 010001010
010001100 010111101 011000010 111011011 111101010 111101100 110010101
110010011 110100010 101110110 101110000 101000001 100111110 011111000
011111110 011001111 010110000 001010011 100011100 111111111 110000000
110110001 110110111 001110001 000001110 000111111 000111001 011101101
011011100 011011010 010100011 010100101 010010100 110001101 111110010
111000011 111000101 100010001 100100000 100100110 101011111 101011001
101101000 010011111 010101110 010101000 011010001 011010111 011100110
000110010 000110100 000000101 001111010 010010101 011101010 011011011
011011101 000001001 000111000 000111110 001000111 001000001 001110000
110000111 110110110 110110000 111001001 111001111 111111110 100101010
100101100 100011101 101100010 001111011 001001010 001001100 000110101
000110011 000000010 011010110 011010000 011100001 010011110 101011000
101011110 101101111 100010000 111110011 011100101 010011010 010101011
010101101 001111001 001001000 001001110 000110111 000110001 000000000
111110111 111000110 111000000 110111001 110111111 110001110 101011010
101011100 101101101 100010010 000001011 000111010 000111100 001000101
001000011 001110010 010100110 010100000 010010001 011101110 100101000
100101110 100011111 101100000 110000011 100010011 100100010 100100100
101011101 101011011 101101010 110111110 110111000 110001001 111110110
000110000 000110110 000000111 001111000 010011011 111001100 111001010
111111011 110000100 101100111 011101001 101000101 100111010 100001011
100001101 111011001 111101000 111101110 110010111 110010001 110100000
001010111 001100110 001100000 000011001 000011111 000101110 011111010
011111100 011001101 010110010 110101011 110011010 110011100 111100101
111100011 111010010 100000110 100000000 100110001 101001110 010001000
010001110 010111111 011000000 000100011 010110011 010000010 010000100
011111101 011111011 011001010 000011110 000011000 000101001 001010110
110010000 110010110 110100111 111011000 100111011 001101100 001101010
001011011 000100100 011000111 101001001 011000011 011110010 011110100
010001101 010001011 010111010 001101110 001101000 001011001 000100110
111100000 111100110 111010111 110101000 101001011 000011100 000011010
000101011 001010100 010110111 100111001 100000100 100000010 100110011
101001100 110101111 000100001 111011101 101100011 101010010 101010100
100101101 100101011 100011010 111001110 111001000 111111001 110000110
001000000 001000110 001110111 000001000 011101011 110111100 110111010
110001011 111110100 100010111 010011001 010100100 010100010 010010011
011101100 000001111 110000001 001111101 011010100 011010010 011100011
010011100 001111111 111110001 000001101 100010101 101110100 101110010
101000011 100111100 111011111 001010001 110101101 010110101 011000101
101100101
```

- All state from demo5

```
000101010   110001010   010000010   011100010   100011110   010010000   001110011
000001100   000111101   000100001   100100010   101000010   010111110   100110000
111010011   110101100   110011101   110000001   001001010   110110110   000111000
011011011   010100100   010010101   010001001   111010110   001011000   010111011
011000100   011110101   011101001   110100100   101000111   100111000   100001001
100010101   011001001   010110110   010000111   010011011   001010101   001100100
001111000   000011011   000000111   000110110   111101010   000010110   110011000
101111011   100000100   100110101   100101001   001110110   111111000   100011011
101100100   101010101   101001001   000000100   011100111   010011000   010101001
010110101   101101001   100010110   100100111   100111011   111110101   111000100
111011000   110111011   110100111   110010110   101111110   011110000   000010011
001101100   001011101   001000001   100001100   111101111   110010000   110100001
110111101   001100001   000011110   000101111   000110011   011111101   011001100
011010000   010110011   010101111   010011110   101101100   110001111   111110000
111000001   111011101   000000001   001111110   001001111   001010011   010011101
010101100   010110000   011010011   011001111   011111110   111111101   110000010
110110011   110101111   101100001   101010000   101001100   100101111   100110011
100000010   011101111   011011110   011000010   010100001   010111101   010001100
001000010   001011110   001101111   000010000   111001100   111100001   000110000
000011101   110111110   110010011   101011101   101110000   100010011   100001111
100111110   001010000   001111101   111011110   111110011   100111101   100010000
101110011   101101111   101011110   000100010   000001111   011000001   011101100
010001111   010010011   010100010   101001111   101100010   100000001   100011101
100101100   111100010   111111110   111001111   110110000   101011000   101110101
011010110   011111011   000110101   000011000   001111011   001100111   001010110
100101010   100000111   111001001   111100100   110000111   110011011   110101010
001000111   001101010   000001001   000010101   000100100   011101010   011110110
011000111   010111000   101001010   101100111   110101001   110000100   111100111
111111011   111001010   000100111   000001010   001101001   001110101   001000100
010001010   010010110   010100111   011011000   111011011   111110110   110010101
110001001   110111000   101110110   101101010   101011011   100100100   011111000
011100100   011010101   010101010   001001001   111000111   000111011   110110101
101010110   100011000   001011011   111010101   100110110   101111000   000101001
011001010   010000100   101000100   100001010   111101001   101010011   011011101
000111110   001110000   100100001   111000010   110001100   001001100   000000010
011100001   101000001   110100010   111101100   000101100   001100010   010000001
111010000   110011110   101111101   011110011   Cannot solve
```

จาก output ดั่งกล่าวที่ Demo5 นั้น แสดงคำว่า Cannot solve นั่นเป็นเพราะว่า graph ที่เราสร้างมานั้นไม่มี node ที่ contain state ที่ไฟดับทั้งหมดไว้ (“000000000”) ซึ่งต่างจาก demo 3,4 ที่ว่าใน graph นั้น contain node ที่ไฟดับทั้งหมดไว้ โดย code ที่ทางคณะผู้จัดทำได้เขียนไว้เพื่อตรวจสอบว่ามี node นั้นหรือไม่คือ code ด้านล่าง

```
        }
        if (!G.containsVertex(finish)) {
            System.out.println("Cannot solve");
        } else {
            sol = shortestPath.getPath(initial, finish).getVertexList();
            showsol(sol);
        }
    }
```

คำสั่ง G.containVertex(finish) มีเพื่อเช็คว่าใน graph ของเรานั้นมี node ที่มี state ที่ไฟดับทั้งหมดไหม ถ้ามีก็จะ return ออก มาเป็น true แล้วก็จะไม่ทำงานใน scope ของ if นั้น แล้วทำการหา shorstest path ใน scope ของ else แต่ถ้าไม่มีก็จะเข้าไปทำงานใน scope ของ if แล้ว print “cannot solve” ซึ่งหมายความว่า puzzle light out จาก initial state นั้น ๆ ไม่สามารถแก้ได้นั่นเอง

Asymptotic runtime ของโปรแกรม

```
32      public void Start() {
33          ShortestPathAlgorithm<Light, DefaultEdge> shortestPath = null;
34          shortestPath = new DijkstraShortestPath<>(G);
35          List<Light> sol = new ArrayList<~>();
36          ArrayList<Light> AL = new ArrayList<~>();
37          Light temp;
38          int i = 0;
39          AL.add(initial);
40          while (i < AL.size()) {
41              temp = AL.get(i);
42              if (!G.containsVertex(temp)) {
43                  G.addVertex(temp);
44              }
45              for (int j = 0; j < allnum; j++) {
46                  String newstate = Totoggle(temp, j);
47                  boolean have = false;
48                  for (Light vertex : G.vertexSet()) {
49                      if (vertex.getpresent().equals(newstate)) {
50                          vertex.setprevious(temp.getpresent(), j);
51                          G.addEdge(temp, vertex);
52                          have = true;
53                      }
54                  }
55                  if (!have) {
56                      Light newlight = new Light(newstate);
57                      G.addVertex(newlight);
58                      AL.add(newlight);
59                      G.addEdge(temp, newlight);
60                  }
61              }
62              i++;
63          }
64          if (!G.containsVertex(finish)) {
65              System.out.println("Cannot solve");
66          } else {
67              sol = shortestPath.getPath(initial, finish).getVertexList();
68              showsol(sol);
69          }
70      }
71
```

จาก Class Game ในส่วนของฟังก์ชัน public void Start() โดย Asymptotic runtime สามารถอธิบายได้ดังนี้

1. ตรง while(i< AL.size()) จากบรรทัดที่ 40 ถึง บรรทัดที่ 63 Asymptotic runtime ของวังวนในกรณี Worst case คือโปรแกรมต้องตรวจทุกปุ่มที่เป็นไปได้ ดังนั้นวังวนนี้จึงมีค่าเป็น $O(2^n)$
2. ตรง if(!G.containsVertex(finish)) จากบรรทัดที่ 64 ถึงบรรทัดที่ 65 กรณี Worst case นั้น ถ้าไม่มีคำตอบจะทำให้วังวนนี้มีค่าเป็น $O(2^n)$
3. ตรง else จากบรรทัดที่ 66 ถึง บรรทัดที่ 69 เป็นการหา ShortestPath โดยใช้วิธีการแบบ Dijkstra ซึ่ง Asymptotic runtime จะมีค่าเป็น $O(|V| + |E|)$ โดยที่ $|V|$ คือจำนวนจุดยอดในกราฟ(V) และ $|E|$ คือจำนวนขอบ(Edge)

โดยสรุปแล้ว การจะหา Asymptotic runtime จาก Worst case ก็คือกรณีที่โปรแกรมไม่สามารถหาคำตอบได้ ซึ่งถ้าหากไม่มีคำตอบ โปรแกรมนี้จะคืนค่า Null ซึ่งจะมี Asymptotic runtime เป็น $O(2^n)$

ข้อจำกัดของโปรแกรม

โปรแกรมไม่สามารถใส่ค่าตั้งต้นได้เกิน 5 (5x5) เพราะจะเกิด “OutOfMemoryError” หรือพื้นที่ความจำระบบเต็ม เนื่องจากโดยทั่วไปตามปกติแล้ว Lights out puzzle สามารถแก้ไขได้สำหรับตารางทุกขนาด แต่ความยากในการไขปริศนาจะเพิ่มขึ้นเมื่อขนาดตารางใหญ่ขึ้น เนื่องจากมีการตั้งค่าสวิตช์ไฟในรูปแบบต่าง ๆ ที่เป็นไปได้หลายแบบให้แก้ไข และการค้นหาค่าที่ถูกต้องที่จะปิดไฟทั้งหมดจะยากขึ้น โดยสาเหตุนี้เป็นผลมาจากโครงสร้างทางคณิตศาสตร์ของปัญหา

บรรณานุกรม

1. Thipwriteblog. 2017. **[Recap] ประเภทของอัลกอริทึม (Types of Algorithm)**. สืบค้น 10 มีนาคม 2566. https://medium.com/thipwriteblog/recap-ประเภทของอัลกอริทึม-types-of-algorithm-a97b1a14044d

2. Quora. 2027. **What is the algorithm for solving lights out puzzle in minimum number of moves in Java?.** สืบค้น 10 เมษายน 2566. https://www.quora.com/What-is-the-algorithm-for-solving-lights-out-puzzle-in-minimum-number-of-moves-in-Java

3. mathworld.wolfram. 2023. **Lights Out Puzzle**. สืบค้น 10 เมษายน 2566. https://mathworld.wolfram.com/LightsOutPuzzle.html

4. Stackexchange. 2013. **Using Dijkstra's algorithm with negative edges?.** สืบค้น 10 เมษายน 2566. https://cs.stackexchange.com/questions/2482/using-dijkstras-algorithm-with-negative-edges

5. Wikipedia. 2023. **Lights Out (Game)**. สืบค้น 10 เมษายน 2566. https://en.wikipedia.org/wiki/Lights_Out_(game)

6. โอภาส เอี่ยมสิริวงศ์. 2016. **โครงสร้างข้อมูล (Data Structures) เพื่อการออกแบบโปรแกรมคอมพิวเตอร์ (ฉบับปรับปรุง)**. ซีเอ็ดยูเคชั่น, บมจ

7. Github **nnichar/Lights-Out-Puzzle** สืบค้น 16 เมษายน 2566 https://github.com/nnichar/Lights-Out-Puzzle/tree/main/src/main/java