Report

EGCO 221 : Data Structure and Algorithms

รศ.ดร รังสิพรรณ มฤคทัต

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

นาย ฉัตริน	มุกสกุล	รหัสนักศึกษา	6213124
นาย วชิรวิชญ์	พีรพิศาลพล	รหัสนักศึกษา	6213145
นาย วิญญ์	ลดาวุฒิพันธ์	รหัสนักศึกษา	6213146

Department of Computer Engineering

Faculty of Engineering, Mahidol University

คำนำ

รายงานฉบับนี้เป็นส่วนหนึ่งของวิชา Data Structure and Algorithms (EGCO 221) โดยคณะ ผู้จัดทำได้จัดทำขึ้นเพื่ออธิบายการทำงานของโปรแกรม Lights Out ซึ่งเป็นโปรแกรมที่ใช้ในการ ตรวจสอบหาวิธีที่ใช้การกดปุ่มปิดไฟน้อยที่สุดเพื่อให้ดวงไฟที่ผู้ใช้ป้อนเข้ามาดับทั้งหมด โดยใช้ Data Structure และ Algorithms มาประยุกต์ใช้ในการแก้ปัญหานี้ โดยในรายงานประกอบไปด้วยคู่มือการใช้งานโปรแกรม การอธิบายการทำงานของ Code และ Algorithms รวมไปถึงข้อจำกัดของโปรแกรม

คณะผู้จัดทำขอขอบพระคุณ รศ.ดร.รังสิพรรณ มฤคทัต ผู้ให้ความรู้และแนวทางในการศึกษา สุดท้าย นี้ทางคณะผู้จัดทำหวังว่ารายงานฉบับนี้ จะให้ความรู้และประโยชน์ไม่มากก็น้อยแก่ผู้อ่านทุกท่าน หากมี ข้อผิดพลาดประการใด ผู้จัดทำจะขอน้อมรับไว้และขออภัยมา ณ ที่นี้ด้วย

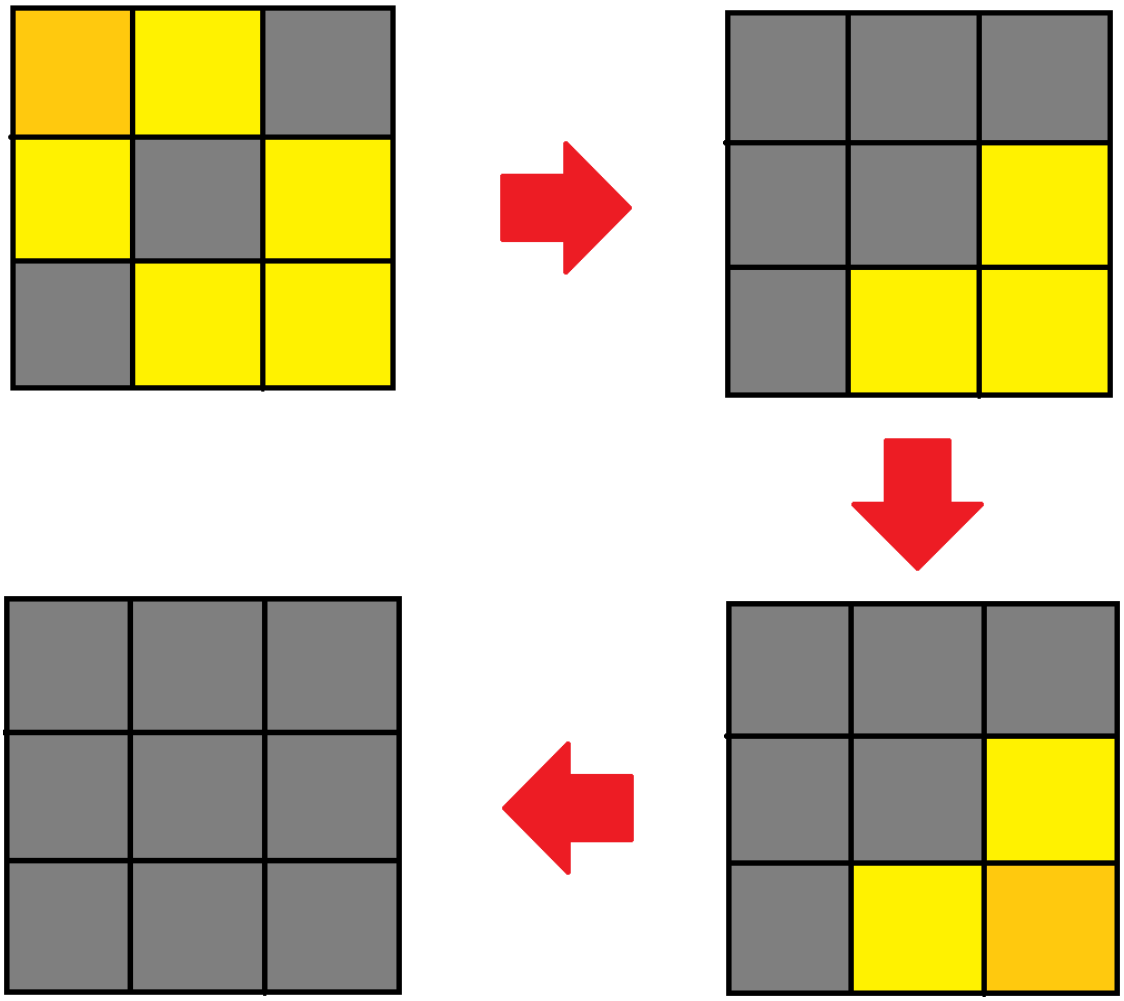
คณะผู้จัดทำ

สารบัญ

หัวข้อ	หน้า
คำนำ	**ก**
สารบัญ	**ข**
เกี่ยวกับโปรแกรม	**1**
คู่มือการใช้งานโปรแกรม (User Manual)	**2**
โครงสร้างข้อมูล (Data Structure)	**4**
- โครงสร้างข้อมูลใน Class State	4
- โครงสร้างข้อมูลใน Class LightOut	5
อัลกอริทึมที่ใช้ในการแก้ปัญหา	**6**
แผนผังการทำงานของโปรแกรม (Flow Chart)	**7**
- แผนผังการทำงานของ Main Function	7
- แผนผังการทำงานของ Constructor ของ Class LightOut	8
- แผนผังการทำงานของ solve Function	9
การทำงานของโปรแกรม	**11**
- การทำงานของ Class State	11
- การทำงานของ Class LightOut	13
ตัวอย่างแสดงการประมวลผล	**17**
Runtime ของโปรแกรม	**22**
ข้อจำกัดของโปรแกรม	**24**
แหล่งอ้างอิง	**25**

เกี่ยวกับโปรแกรม

โปรแกรมนี้เป็นโปรแกรมที่ใช้สำหรับการแก้ปัญหาการปิดไฟ โดยจะรับขนาดตารางจำนวน N ที่ต้องมีขนาดไม่น้อยกว่า 3 ขึ้นไปกับค่าสถานะไฟ จำนวน N x N และจะแสดงผลลัพธ์ลำดับการปิดไฟที่เป็นไปได้ทั้งหมด โดยมีเงื่อนไขว่าเมื่อมีการเลือกเลือกไฟที่จุดไหนนั้นจะทำการเปลี่ยนสถานะ และไฟรอบๆต้องทำการสลับค่าสถานะไฟด้วยเช่นกันไม่ว่าจะด้านซ้าย ด้านขวา ด้านบนและด้านล่าง และให้ได้ผลลัพธ์คือสถานะไฟเป็นไฟปิดทั้งหมดดังตัวอย่างต่อไปนี้



หมายเหตุ :
- คือ ปุ่มที่กด
- คือ ปุ่มที่ไฟติด
- คือ ปุ่มที่ไฟดับ

คู่มือการใช้งานโปรแกรม (User manual)

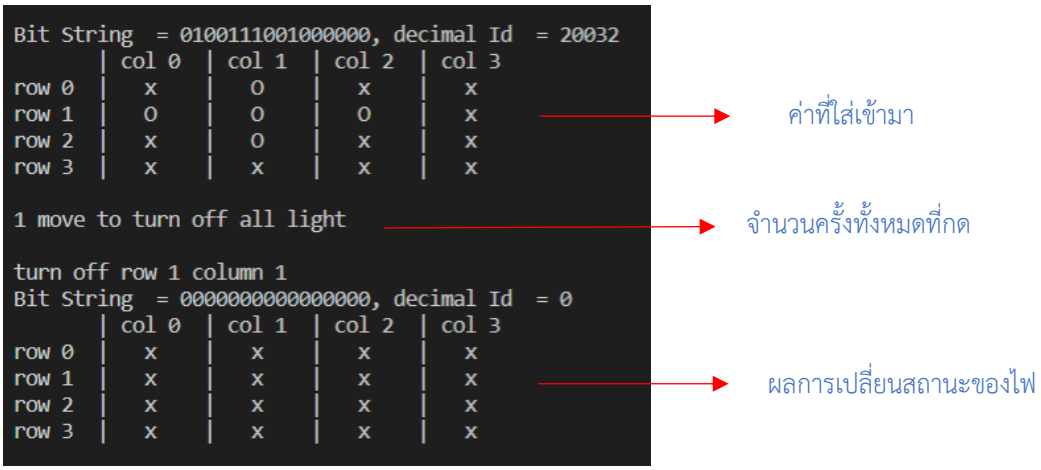
1. เมื่อโปรแกรมเริ่มทำงาน จะให้ User ป้อนค่าขนาดของตารางไฟที่เป็นตารางรูปสี่เหลี่ยมจตุรัส โดยโปรแกรมกำหนดไว้ว่าความยาวด้านของตารางต้องเป็น 3 หรือ 4 เท่านั้น และขนาดที่ป้อนเข้ามานั้นจะต้องเป็นตัวเลขจำนวนเต็ม

```
Number of rows for sqaure grid(3-4) =
1
Grid size must be 3 or 4
Number of rows for sqaure grid(3-4) =
-1
Grid size must be 3 or 4
Number of rows for sqaure grid(3-4) =
5
Grid size must be 3 or 4
Number of rows for sqaure grid(3-4) =
three
java.util.InputMismatchException
Number of rows for sqaure grid(3-4) =
4
```

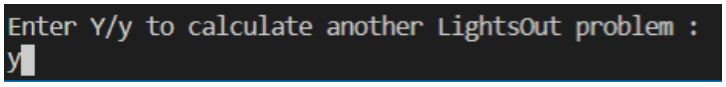
2. จากนั้นให้ User ทำการป้อนค่าสถานะของไฟแต่ละตำแหน่ง โดยโปรแกรมกำหนดไว้ว่าค่าสถานะของไฟที่รับค่าเข้ามาจะต้องมีค่าเป็นเลข 0 (ไฟดับ) และ 1 (ไฟติด) จำนวนค่าสถานะของไฟที่ป้อนเข้ามาจะต้องเป็นตัวเลขและจะต้องมีจำนวนตัวเลขเท่ากับขนาดที่ใส่เข้าไปยกกำลังสอง

```
Initial states (16 bits , left to right ,line by line ) =
22222222222
Initial states have only 0 or 1
Initial states (16 bits , left to right ,line by line ) =
44444444444444444444444444
Initial states have only 0 or 1
Initial states (16 bits , left to right ,line by line ) =
11111
current length = 5
Initial states (16 bits , left to right ,line by line ) =
2222
Initial states have only 0 or 1
Initial states (16 bits , left to right ,line by line ) =
11111111
current length = 8
Initial states (16 bits , left to right ,line by line ) =
00000000011111
current length = 14
Initial states (16 bits , left to right ,line by line ) =
1111111111111100
```

3. หลังจากที่กรอก ข้อมูลขนาดตารางและค่าสถานะของไฟในแต่ละตำแหน่งแล้ว โปรแกรมจะทำการคำนวณวิธีการกดไฟให้ไฟนั้นดับทั้งหมดโดยจะบอกเป็นแถวและหลักของตาราง พร้อมกับแสดงจำนวนการกดทั้งหมด



4. เมื่อโปรแกรมจบการแสดงวิธีการปิดไฟแล้วโปรแกรมนั้นจะมีข้อความให้ใส่ว่าจะใส่ปัญหาแบบอื่นอีกหรือไม่ ถ้าใส่ Y/y จะทำการให้ใส่ข้อมูลใหม่อีกรอบ



โครงสร้างข้อมูล (Data Structure)

1. โครงสร้างข้อมูลใน class State

```
class State
{
    private boolean [][]light;
    private int lastPress;

    public State(boolean [][] s, int l)
    {
        light = s;
        lastPress = l;
    }

    public boolean[][] getState()
    {
        return light;
    }

    public int getLastPress()
    {
        return lastPress;
    }

    public boolean[][] toggle(int n)
    {...30 lines }

    @Override
    public String toString()
    {...13 lines }
}
```

Class State เป็น class ที่ทำหน้าที่เก็บค่า State ของไฟในแต่ละรูปแบบเอาไว้ในรูปแบบของ Array ของ Boolean และเก็บค่าการกดครั้งล่าสุดของ State นั้นๆเอาไว้ในรูปแบบของ int ประกอบด้วย method เพื่อใช้ในการ return ค่าของตัวแปรใน class ได้แก่ getState() และ getLastPress() มี method toggle(int) สำหรับเปลี่ยน State ของไฟในแต่ละรูปแบบ และ override method toString()

2. โครงสร้างข้อมูลใน class LightOut (main class)

```
public class LightOut
{
    private int n_size=0;
    private String str_start, ans = "";
    private boolean [][]light;

    public LightOut()
    {...63 lines }

    public void solve()
    {...58 lines }

    public void changeState(String str_bits)
    {...33 lines }

    public void print(String bit)
    {...18 lines }

    public static void main(String argv[])
    {
        String run;
        do
        {
            new LightOut();
            System.out.println("Enter Y/y to calculate another LightsOut problem : ");
            Scanner in = new Scanner(System.in);
            run = in.nextLine();
        }while(run.equalsIgnoreCase("y"));
    }
}
```

Class LightOut ทำหน้าที่เป็น main class ของโปรแกรม ทำหน้าที่รับค่าจากผู้ใช้ ประกอบด้วยตัวแปร n_size คือขนาดของปัญหา str_start คือรูปแบบ String ของ state เริ่มต้นของไฟของปัญหา ans คือ Null string ซึ่งจะกลายเป็นรูปแบบ String ของ state ของไฟที่ดับทั้งหมด และ Array boolean 2 มิติ light คือตัวแปรเก็บรูปแบบ state ของไฟ และมี 3 method ได้แก่ solve() ซึ่งเป็น method หลักในการหาคำตอบของปัญหา changeState(String) เป็น method เพื่อเปลี่ยน state ของรูปแบบของ output และ print(String) ทำหน้าที่แสดงผลของไฟในแต่ละรูปแบบ

อัลกอริทึมที่ใช้ในการแก้ปัญหา

การแก้ปัญหานี้ เราเลือกใช้ ArrayDeque ในการเก็บข้อมูล เนื่องจากสามารถดึงค่าและลบข้อมูลของตัวแรกและตัวสุดท้ายของชุดข้อมูลได้ โดยมีตัวแปร bfs ซึ่งเป็น ArrayDeque ที่ใช้ในการเก็บ Object State ที่มีข้อมูลรูปแบบไฟและปุ่มกดล่าสุด เพื่อนำมาหารูปแบบที่เป็นไปได้ โดยวิธีการ Breadth-first Search แล้วนำรูปแบบไปสร้างกราฟ

ขั้นตอนการประมวลผล

1. สร้าง State ที่มีรูปแบบเป็น Initial (ข้อมูลที่รับจาก User) ที่มีปุ่มที่กดล่าสุดเป็น 0 (ปุ่มที่กดล่าสุดไม่ใช่จำนวนปุ่มทั้งหมดที่กด) เพิ่มเข้าในตัวแปร bfs

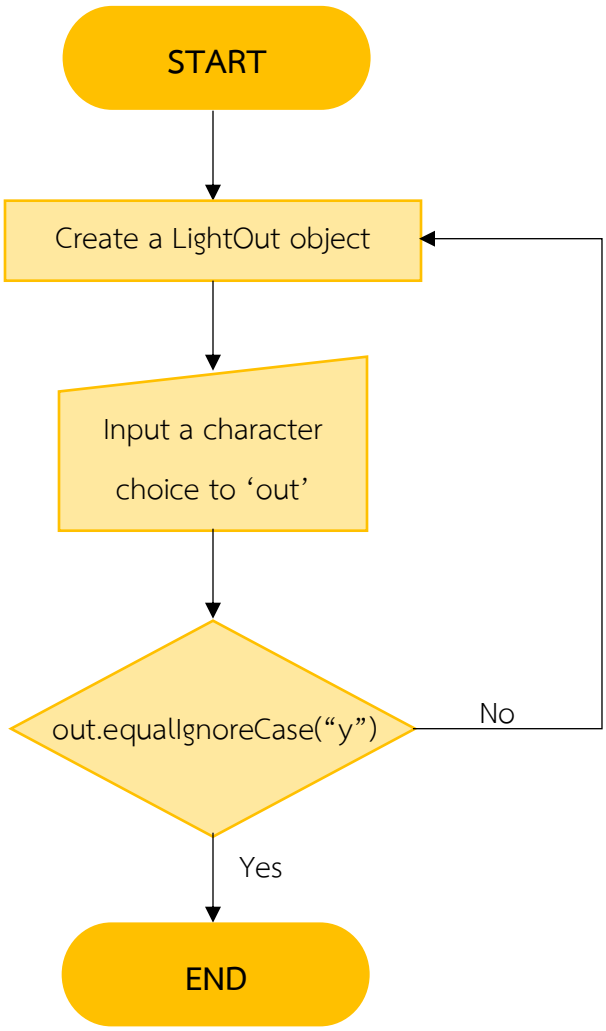
2. ดึงข้อมูลตัวแรกสุดออก มาแล้วนำไปหารูปแบบต่อไปที่เป็นไปได้ทั้งหมด โดยปุ่มที่ถูกกดจะไล่ตั้งแต่ปุ่มล่าสุด + 1 ตลอดจนถึง ปุ่มสุดท้าย เช่น ตาราง 3*3 จะมีปุ่มที่กดได้ตั้งแต่ 1 ถึง 9 แล้วสร้างโหนดในกราฟพร้อมกับเส้นเชื่อมของตัวแรกกับรูปแบบใหม่ที่หาได้ ถ้ายังไม่มีโหนดของรูปแบบนั้นในกราฟให้เพิ่มเข้า bfs เพื่อลดระยะเวลาจากการทำงานซ้ำรูปแบบเดิม

3. ทำข้อ 2 ซ้ำจนกว่าจะเจอคำตอบ หรือจนกว่าข้อมูลใน bfs ถูกดึงมาจนหมด ซึ่งหมายความว่าไม่สามารถหาคำตอบได้ แล้วจะข้ามการทำงานในลำดับที่ 4 ไป

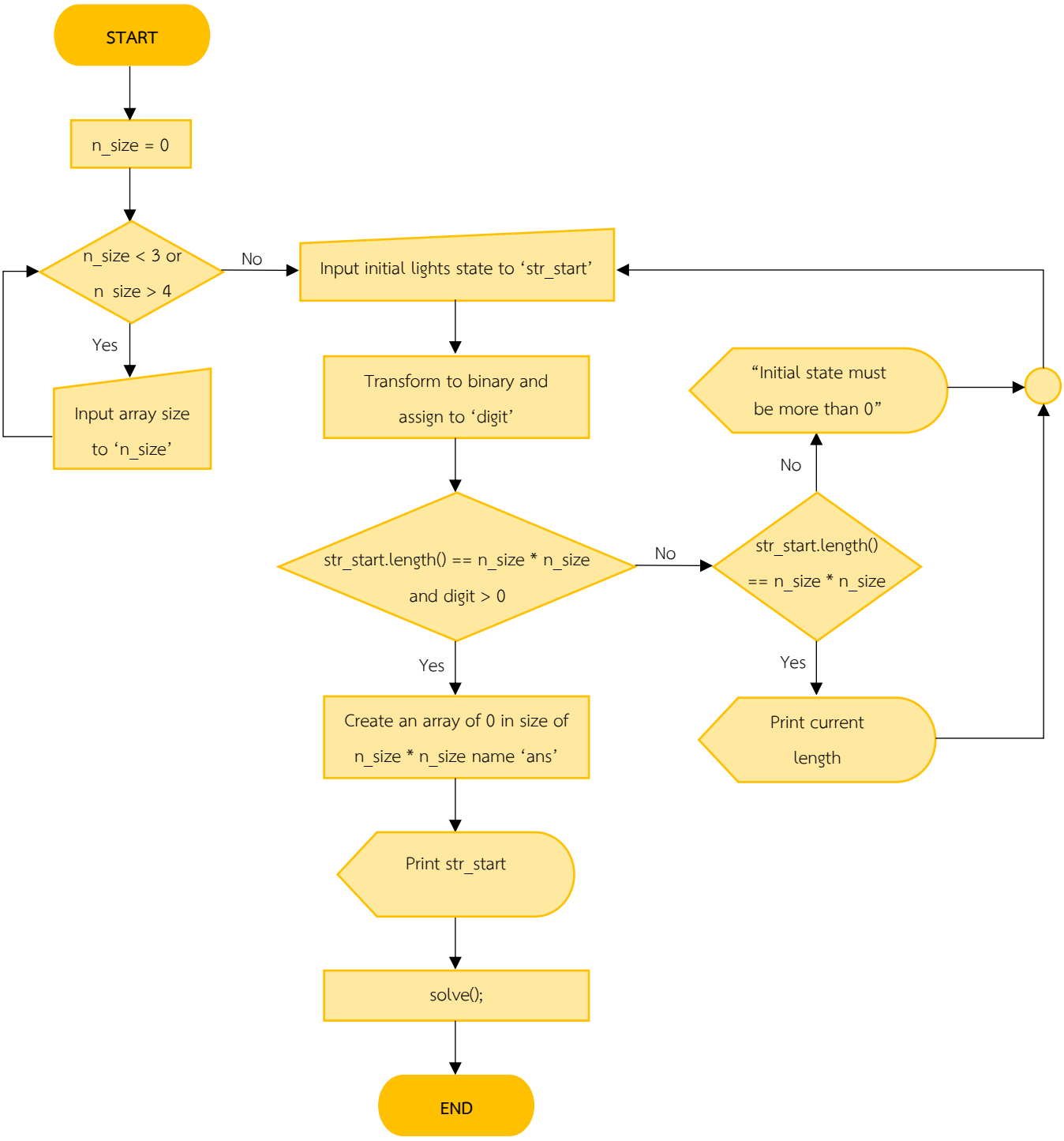
4. เมื่อทำขั้นตอนหารูปแบบทั้งหมดจากการทำ Breadth-first Search แล้วพบว่าสามารถหาคำตอบได้ จะทำการหา Shortest Path ด้วยการใช้ Graph API โดยเลือกใช้ DijkstraShortestPath เนื่องจากแต่ละเส้นเชื่อมไม่มีค่าติดลบ Dijkstra’s algorithm จึงเหมาะสมที่สุด

แผนผังการทำงานของโปรแกรม (Flow chart)

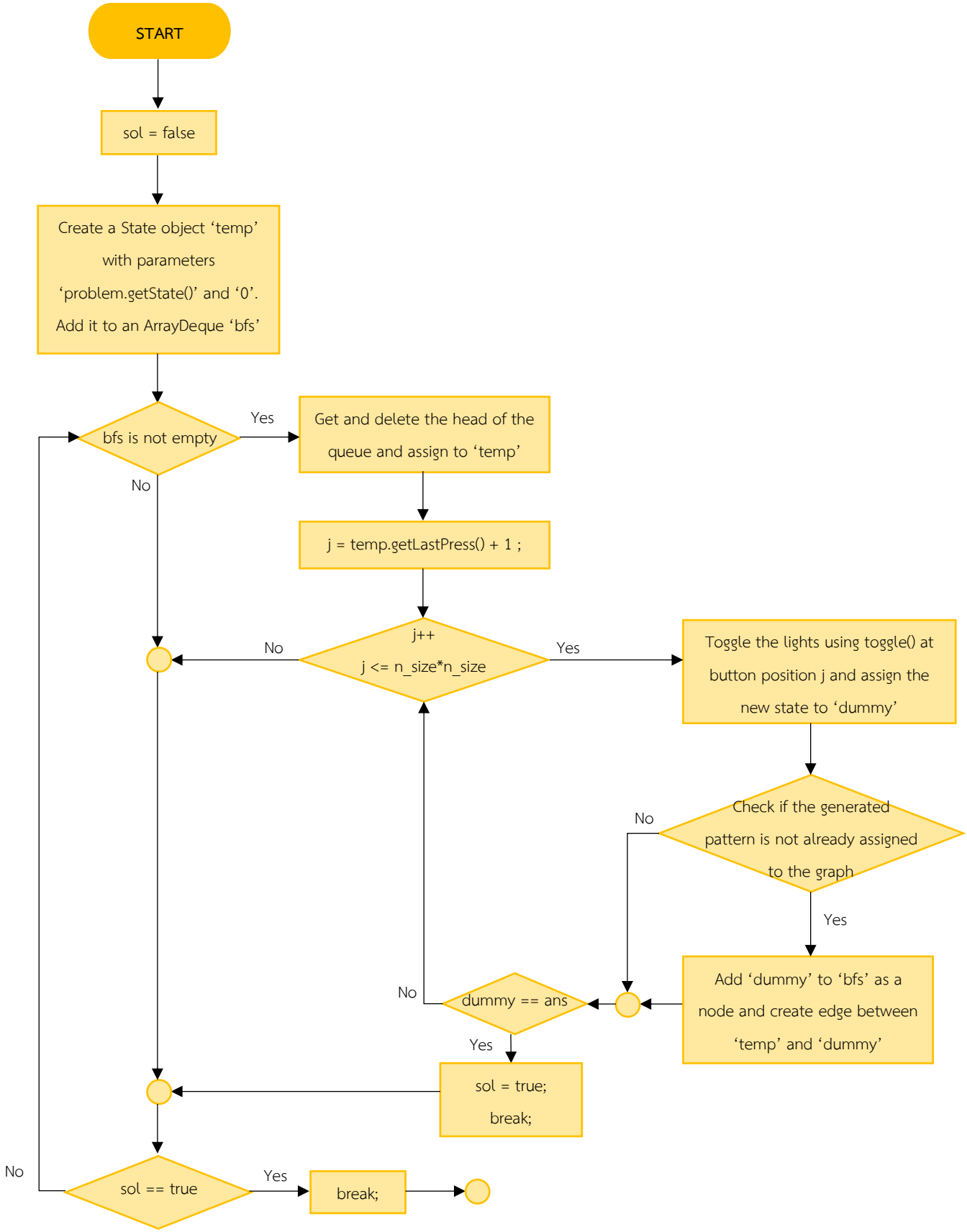
1. แผนผังการทำงานในฟังก์ชัน Main



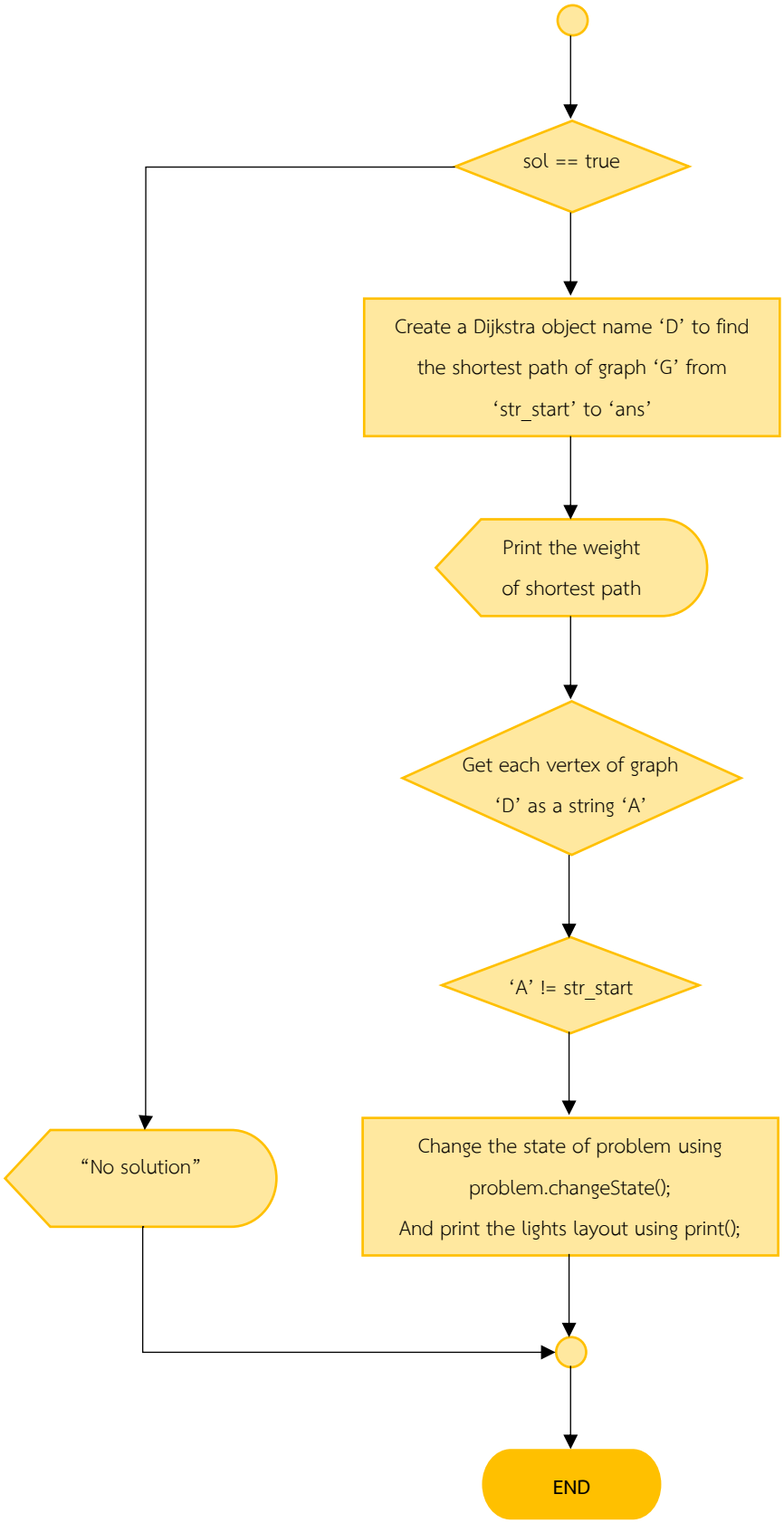
2. แผนผังการทำงานใน Constructor ของ LightOut



3. แผนผังการทำงานในฟังก์ชัน solve



3. แผนผังการทำงานในฟังก์ชัน solve (ต่อ)



การทำงานของโปรแกรม

1. Class State เก็บรูปแบบของไฟและปุ่มล่าสุดที่ถูกกด ไว้คำนวณหารูปแบบทั้งหมด

```
class State
{
    private boolean [][]light;
    private int lastPress;

    public State(boolean [][] s, int l)
    {
        light = s;
        lastPress = l;
    }

    public boolean[][] getState()
    {
        return light;
    }

    public int getLastPress()
    {
        return lastPress;
    }


    @Override
    public String toString()
    {
        String re="";

        for(int i=0;i<light.length;i++)
        {
            for(int j=0;j<light.length;j++)
            {
                re = re + (light[i][j]==true?'1':'0');
            }
        }

        return re;
    }
}
```

Constructor รับ array boolean 2 มิติ ที่เป็นรูปแบบไฟ และ int เป็นปุ่มล่าสุด

Functions คืนค่าตัวแปรของ State

```
public boolean[][] toggle(int n)
{
    int n_size = light.length;

    boolean[][] intTemp=new boolean[light.length][light.length];

    for(int i=0;i<n_size;i++)
    {
        for(int j=0;j<n_size;j++)
        {
            intTemp[i][j] = light[i][j];
        }
    }

    int row = (n-1)/n_size, column = (n-1)%n_size;

    intTemp[row][column] = intTemp[row][column]==true?false:true;
    if(row-1>-1)//up
        intTemp[row-1][column] = intTemp[row-1][column]==true?false:true;

    if(row+1<n_size)//down
        intTemp[row+1][column] = intTemp[row+1][column]==true?false:true;

    if(column-1>-1)//left
        intTemp[row][column-1] = intTemp[row][column-1]==true?false:true;

    if(column+1<n_size)//right
        intTemp[row][column+1] = intTemp[row][column+1]==true?false:true;

    return intTemp;
}
```

คัดลอกค่า

Function เปลี่ยนค่าหลอดไฟแต่ละหลอด เมื่อกดหลอดไฟหลอดที่ n

2. Class LightOut (Main Class) รับค่าและคำนวณหารูปแบบการกดไฟที่น้อยที่สุด

```
public class LightOut
{
    private int n_size=0;
    private String str_start, ans = "";
    private boolean [][]light;

    public LightOut()
    {
        while(n_size<3||n_size>4)
        {
            try
            {
                System.out.println("Number of rows for sqaure grid(3-4) = ");
                Scanner input = new Scanner(System.in);
                n_size = input.nextInt();
                if(n_size<3||n_size>4)
                    System.out.println("Grid size must be 3 or 4");
            }
            catch(Exception ex)
            {
                System.err.println(ex);
            }
        }

        while(true)
        {
            Scanner input2 = new Scanner(System.in);
            try
            {
                System.out.printf("Initial states (%d bits , left to right ,line by line ) = \n", n_size*n_size);
                str_start  = input2.nextLine();
                int digit = Integer.parseInt(str_start, 2);
                if(str_start.length()==n_size*n_size&&digit>0)
                {
                    for(int i=0;i<n_size*n_size;i++)
                    {
                        ans = ans+"0";
                    }
                    break;
                }

                if(str_start.length()!=n_size*n_size)
                    System.out.println("current length = " + str_start.length());
                else
                    System.out.println("Initial states must be more than 0");
            }
            catch(NumberFormatException e)
            {
                System.err.println("Initial states have only 0 or 1");
            }
            catch(Exception ex)
            {
                System.out.println(ex);
            }
        }

        light = new boolean[n_size][n_size];

        int temp=0;
        for(int i=0;i<n_size;i++)
        {
            for(int j=0;j<n_size;j++, temp++)
            {
                light[i][j] = str_start.charAt(temp)=='1'?true:false;
            }
        }

        print(str_start);
        solve();
    }
```

Constructor

รับและตรวจสอบข้อมูลต่างๆ

แล้วเรียก solve() เพื่อหาคำตอบ

รับข้อมูลขนาดของ Grid

ต้องเป็น 3 หรือ 4 เท่านั้น

รับข้อมูลรูปแบบโจทย์

ใส่ได้แค่คำที่ประกอบด้วยเลข 0, 1

ต้องมีจำนวนตัวอักษรตรงตามจำนวน

หลอดไฟและค่า digit มากกว่า 0

สร้างรูปแบบไฟที่เป็นเมทริกซ์

```
public void solve()
{
    boolean sol=false;
    Graph<String, DefaultEdge> G = new SimpleGraph(DefaultEdge.class);
    ArrayDeque<State> bfs = null;
    try
    {
        State temp = new State(light, 0);
        bfs = new ArrayDeque<State>();

        bfs.add(temp);
        while(!bfs.isEmpty())
        {
            temp = bfs.pollFirst();
            for(int j = temp.getLastPress()+1;j <= n_size*n_size;j++)
            {
                State dummy = new State(temp.toggle(j), j);
                if(!G.containsVertex(dummy.toString()))
                {

                    bfs.add(dummy);
                }

                Graphs.addEdgeWithVertices(G, temp.toString(), dummy.toString());

                if(dummy.toString().equals(ans))
                {
                    sol=true;
                    break;
                }
            }
            if(sol)
                break;
        }
    }
    catch(Exception ex)
    {
        sol = false;
        System.out.println(ex);
    }
    if(sol)
    {
        DijkstraShortestPath<String, DefaultEdge> D = new DijkstraShortestPath(G);
        System.out.printf("%.0f move to turn off all light\n\n", D.getPath(str_start, ans).getWeight());
        List<String> a = D.getPath(str_start, ans).getVertexList();
        a.remove(0);
        for(String A : a)
        {
            changeState(A);
            print(A);
        }
    }
    else
    {
        System.out.println("No Solution");
    }
}
```

สร้างกราฟเพื่อใช้เก็บทุกรูปแบบที่หาได้

ArrayDeque เก็บตัวเข้าคิวเพื่อหารูปแบบ

ดึงค่าตามคิว

หารูปแบบจากการกดปุ่ม j ถ้าไม่เคยมี

รูปแบบนี้ให้เพิ่มเข้าคิวเพื่อใช้หารูปแบบต่อไป

สร้างเส้นเชื่อมระหว่างรูปแบบที่นำมา

คิดกับรูปแบบใหม่ที่หาได้

จบลูปด้วยส่วนนี้

แสดงว่าหาคำตอบได้

ส่วนที่ใช้หาว่ามีคำตอบหรือไม่

โดยใช้หลักการ Breadth-first Search

จบลูปด้วยส่วนนี้

ไม่สามารถหาคำตอบได้

หา Shortest Path แล้วดึงค่า โหนด แต่ละ

โหนดที่เป็นคำตอบเพื่อนำไปเปลี่ยนรูปแบบ

โดยใช้ changeState แล้วแสดงผล

```java
public void changeState(String str_bits)
{
    ArrayList<int[]> change = new ArrayList();
    int temp=0;
    for(int i=0;i<n_size;i++)
    {
        for(int j=0;j<n_size;j++, temp++)
        {
            if(!((str_bits.charAt(temp)=='1'?true:false)==light[i][j]))
            {
                light[i][j] = !light[i][j];
                change.add(new int[]{i, j});
            }
        }
    }
    boolean check;
    for(int[]n:change)
    {
        check = true;
        for(int[]m:change)
        {
            if(Math.abs(n[0]-m[0])+Math.abs(n[1]-m[1])!=1 && m!=n)
            {
                check = false;
                break;
            }
        }
        if(check)
        {
            System.out.printf("%s row %d column %d",light[n[0]][n[1]]==true?"turn on":"turn off", n[0], n[1]);
            break;
        }
    }
}
```

คำนวณหาตำแหน่งใดเปลี่ยนค่าแล้วเก็บใส่ change ที่เป็น ArrayList<int[]>

เก็บตำแหน่งหลอดไฟที่มีการเปลี่ยนค่า

คำนวณหาตำแหน่งที่ถูกกด

Function เปลี่ยนรูปแบบไฟ
คำนวณหาปุ่มที่ถูกกด
แล้วแสดงผลการคำนวณ

แสดงผลตำแหน่งที่ถูกกดว่าเปิดหรือปิด

```java
public void print(String bit)
{
    System.out.printf("\nBit String  = %s, decimal Id  = %d\n      ", bit, Integer.parseInt(bit, 2));
    for(int i=0;i<n_size;i++)
    {
        System.out.printf(" | col %d ", i);
    }
    System.out.println("");
    for(int i=0;i<n_size;i++)
    {
        System.out.printf("row %d ", i);
        for(int j=0;j<n_size;j++)
        {
            System.out.printf(" |   %c   ", (light[i][j]==true)?'O':'x');
        }
        System.out.println("");
    }
    System.out.println("");
}
```

แสดงผลลัพธ์

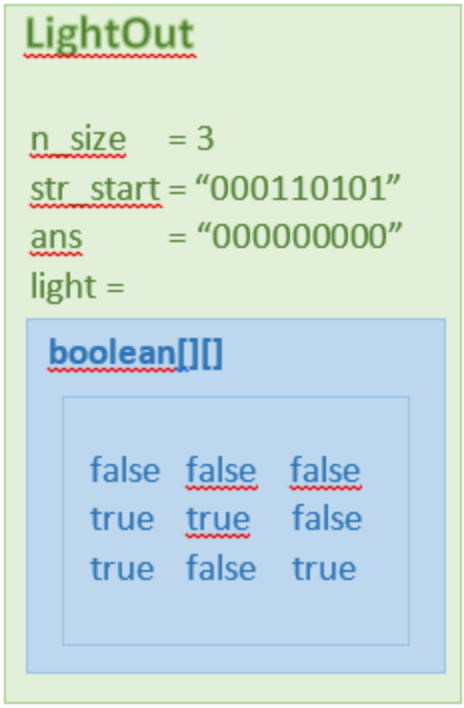
```
public static void main(String argv[])
{
    String run;
    do
    {
        new LightOut();
        System.out.println("Enter Y/y to calculate another LightsOut problem : ");
        Scanner in = new Scanner(System.in);
        run = in.nextLine();
    }while(run.equalsIgnoreCase("y"));
}
```

ใน main จะทำการเรียก LightOut เปรียบเสมือนการทำงานเมื่อเสร็จสิ้นแล้วจึงถาม User ว่าต้องการหาคำตอบโจทย์ LightOut อื่นอีกหรือไม่โดยไม่ต้องรันโปรแกรมใหม่ โดยหาก User ใส่ตัวอักษร Y หรือ y จะทำการเรียก LightOut และเริ่มโปรแกรมใหม่อีกครั้งหนึ่ง

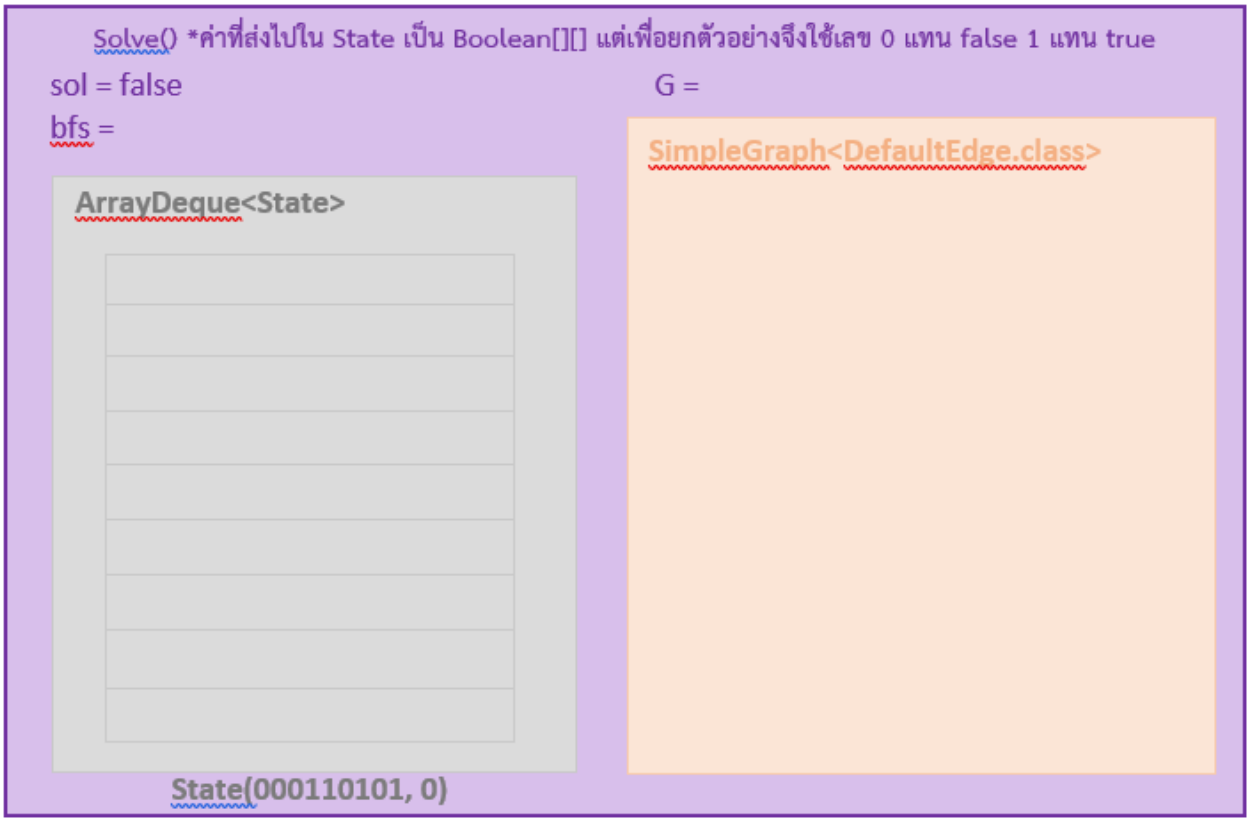
ตัวอย่างแสดงการประมวลผล

```
Number of rows for sqaure grid(3-4) =
3
Initial states (9 bits , left to right ,line by line ) =
000110101

Bit String  = 000110101, decimal Id  = 53
        | col 0  | col 1  | col 2
row 0   |   x    |   x    |   x
row 1   |   O    |   O    |   x
row 2   |   O    |   x    |   O
```



รับค่า grid ใส่ใน n_size แล้วนำ n_size มาคำนวณหาจำนวนตัวอักษร รับตัวอักษรเก็บใส่ str_start แล้วเมื่อค่าของ str_start ถูกต้องค่อยสร้าง ans แล้วจึงสร้าง problem ที่เป็น Object Light หลังจากนั้นจึงเข้าสู่การคำนวณ โดยเรียก solve() ที่อยู่ใน LightOut



Solve() *เข้า while loop ครั้งแรก ดึงตัวแรกใน bfs ออก (State(000110101, 0)) เพื่อนำมาคิด

sol = false

bfs =

ArrayDeque<State>

State(000111110, 9)
State(000100010, 8)
State(000010011, 7)
State(001101100, 6)
State(010001111, 5)
State(100000001, 4)
State(011111101, 3)
State(111100101, 2)
State(110010101, 1)

G =

SimpleGraph<DefaultEdge.class>

Solve() *เข้า while loop ครั้ง 2 ดึง (State(110010101, 1)) เพื่อนำมาคิด

sol = false

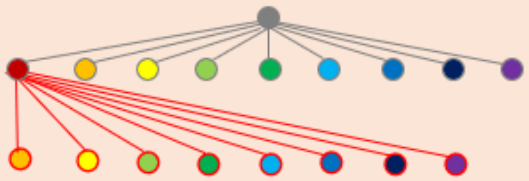
bfs =

ArrayDeque<State>

State(110011110, 9)
:
State(100101111, 5)
State(010100001, 4)
State(101011101, 3)
State(001000101, 2)
State(000111110, 9)
:
State(111100101, 2)

G =

SimpleGraph<DefaultEdge.class>



Solve() *เข้า while loop ครั้ง 3 ถึง (State(111100101, 2)) เพื่อนำมาคิด

sol = false

bfs =

ArrayDeque<State>

State(111101110, 9)
:
State(110111100, 6)
State(101011111, 5)
State(011010001, 4)
State(100101101, 3)
State(110011110, 9)
:
State(011111101, 3)

G =

SimpleGraph<DefaultEdge.class>

Solve() *เข้า while loop จนเจอคำตอบ (State(000000000, ?))

sol = true

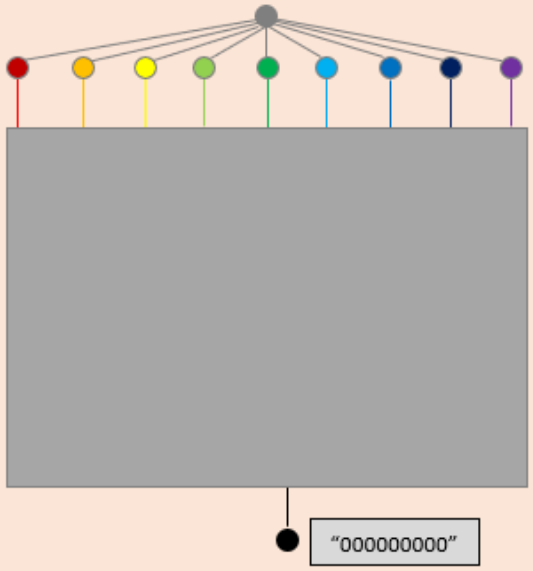
bfs =

ArrayDeque<State>

State(000000000, ?)
:
:
:
:
:
:
:
:

G =

SimpleGraph<DefaultEdge.class>



เมื่อเช็คว่ามีคำตอบจากการทำ Breadth-first Search แล้วสร้าง DijkstraShortestPath เพื่อนำมาหาเส้นทางที่สั้นที่สุดของกราฟ G ที่เราได้สร้างขึ้นมา

เลข 5 ได้จาก .getPath(str_strat, ans).getWeight()

ของ DijkstraShortestPath

```
5 move to turn off all light
```

```
turn on row 0 column 0
Bit String  = 110010101, decimal Id  = 405
       | col 0  | col 1  | col 2
row 0  |   O    |   O    |   x
row 1  |   x    |   O    |   x
row 2  |   O    |   x    |   O
```

change ArrayList<int[]> { {0, 0}, {0, 1}, {1, 0} }

```
turn off row 0 column 1
Bit String  = 001000101, decimal Id  = 69
       | col 0  | col 1  | col 2
row 0  |   x    |   x    |   O
row 1  |   x    |   x    |   x
row 2  |   O    |   x    |   O
```

change ArrayList<int[]> { {0, 0}, {0, 1}, {0, 2}, {1, 1} }

```
turn on row 1 column 2
Bit String  = 000011100, decimal Id  = 28
       | col 0  | col 1  | col 2
row 0  |   x    |   x    |   x
row 1  |   x    |   O    |   O
row 2  |   O    |   x    |   x
```

change ArrayList<int[]> { {0, 2}, {1, 1}, {1, 2}, {2, 2} }

เปลี่ยนข้อมูล light โดยใช้ changeState

LightOut
n_size = 3
str_start = “000110101”
ans = “000000000”
light =
boolean[][]

true	true	false
false	true	false
true	false	true

LightOut
n_size = 3
str_start = “000110101”
ans = “000000000”
light =
boolean[][]

false	false	true
false	false	false
true	false	true

LightOut
n_size = 3
str_start = “000110101”
ans = “000000000”
light =
boolean[][]

false	false	false
false	true	true
true	false	false

LightOut
n_size = 3
str_start = “000110101”
ans = “000000000”
light =

boolean[][]

false	false	false
false	false	true
false	true	true

```
turn on row 2 column 1
Bit String  = 000001011, decimal Id  = 11
       | col 0  | col 1  | col 2
row 0  |   x    |   x    |   x
row 1  |   x    |   x    |   O
row 2  |   x    |   O    |   O
```

change ArrayList<int[]> { {1, 1}, {2, 0}, {2, 1}, {2, 2} }

LightOut
n_size = 3
str_start = “000110101”
ans = “000000000”
light =

boolean[][]

false	false	false
false	false	false
false	false	false

```
turn off row 2 column 2
Bit String  = 000000000, decimal Id  = 0
       | col 0  | col 1  | col 2
row 0  |   x    |   x    |   x
row 1  |   x    |   x    |   x
row 2  |   x    |   x    |   x
```

change ArrayList<int[]> { {1, 2}, {2, 1}, {2, 2} }

```
Enter Y/y to calculate another LightsOut problem :
0
```

ถ้าใส่ตัวอักษร Y หรือ y โปรแกรมจะเริ่มรันให้ใส่ค่าใหม่ หรือใส่ตัวอักษรอื่นจะจบโปรแกรมทันที

Runtime ของโปรแกรม

```java
public void solve()
{
    boolean sol=false;
    Graph<String, DefaultEdge> G = new SimpleGraph(DefaultEdge.class);
    ArrayDeque<State> bfs = null;
    try
    {
        State temp = new State(problem.getState(), 0);
        bfs = new ArrayDeque<State>();

        bfs.add(temp);
        while(!bfs.isEmpty())
        {
            temp = bfs.pollFirst();
            for(int j = temp.getLastPress()+1;j <= n_size*n_size;j++)
            {
                State dummy = new State(temp.toggle(j), j);
                if(!G.containsVertex(dummy.toString()))
                {

                    bfs.add(dummy);
                }

                Graphs.addEdgeWithVertices(G, temp.toString(), dummy.toString());

                if(dummy.toString().equals(ans))
                {
                    sol=true;
                    break;
                }
            }
            if(sol)
                break;

        }
    }
    catch(Exception ex)
    {
        sol = false;
        System.out.println(ex);
    }

    if(sol)
    {
        DijkstraShortestPath<String, DefaultEdge> D = new DijkstraShortestPath(G);
        System.out.printf("%.0f move to turn off all light\n\n", D.getPath(str_start, ans).getWeight());
        for(String A : D.getPath(str_start, ans).getVertexList())
        {
            if(!A.equals(str_start))
            {
                problem.changeState(A);
                problem.print();
            }
        }
    }
    else
    {
        System.out.println("No Solution");
    }
}
```

Asymptotic runtime ในช่วงของการเพิ่มค่าลงกราฟนั้น มีruntime ในกรณีแย่สุดคือ $O(2^{N^2})$

Algorithm นี้ ดีกว่าการ Brute-force อย่างไร

Asymptotic runtime ในช่วงของการเพิ่มค่ารูปแบบสถานะไฟที่เป็นไปได้ลงกราฟโดยใช้วิธีแบบ Breadth first search นั้นจะมี asymptotic runtime ในกรณีแย่สุดคือ O(2^{N^2}) จึงดีกว่าการทำแบบ brute forceที่ได้ asymptotic runtime เป็น O(2^{N^2}) เสมอ เนื่องจากได้มีการเช็คว่าถ้ารูปแบบของสถานะไฟนี้มีอยู่แล้วจะไม่นำเข้าqueueเพื่อไปทำใหม่อีกรอบ

ข้อจำกัดของโปรแกรม

โปรแกรมนี้สามารถใช้ในการแก้ปัญหาได้เฉพาะขนาด 3×3 และ 4×4 เท่านั้น เนื่องจากหากปัญหามีขนาดมากกว่า 4×4 จำนวนรูปแบบค่าสถานะไฟที่เป็นไปได้ที่โปรแกรมคิดได้ จะมีขนาดเยอะมากจนทำให้โปรแกรมค้างและไม่สามารถทำงานต่อได้

แหล่งอ้างอิง

1. Zulaikha Lateef. All You Need To Know About The Breadth First Search Algorithm. [ออนไลน์]. (วันที่ค้นหาข้อมูล : 24 เมษายน 2564). เข้าถึงได้จาก www.edureka.co/blog/breadth-first-search-algorithm/

2. Rafael Losada Liste. Lights Out (Games with solutions). [ออนไลน์]. (วันที่ค้นหาข้อมูล : 24 เมษายน 2564) เข้าถึงได้จาก www.geogebra.org/m/JexnDJpt