EGCO 221 – Group Project 2 (Light Out Puzzle)

*** The project can be done in a group of <=4 students. But each group must do it by themselves. **Everyone involved in cheating will get ZERO point**

1. Lights are placed in N x N grid. Each of them can be ON (1) or OFF (0).
   1.1 When a normal light is toggled (ON➔OFF or OFF➔ON), so are its horizontal and vertical neighbors, if any of them exists. For example,

```
0  0  0       0  1  0                0  0  0       1  1  0
1  1  1  ->   0  0  0                1  1  1  ->   0  1  1
1  0  1       1  1  1                1  0  1       1  0  1
```

   light at the middle is toggled — *light at the top-left is toggled*

   1.2 There may be 1 broken light on the grid (as marked by 'x'). When the broken light is toggled, its diagonal neighbors will be toggled instead. For example,

```
0  0  0       1  0  1                0x 0  0       1x 0  0
1  1x 1  ->   1  0x 1                1  1  1  ->   1  0  1
1  0  1       0  0  0                1  0  1       1  0  1
```

   broken light at the middle is toggled — *broken light at the top-left is toggled*

   Note that the diagonal effect occurs only when we toggle the broken light. After the above step, if we toggle a normal light, the effect will be as in (1.1). For example,

```
1  0  1       0  1  1                1x 0  0       1x 1  0
1  0x 1  ->   0  0x 1                1  0  1  ->   0  1  0
0  0  0       0  0  0                1  0  1       1  1  1
```

   light at the top-left is toggled — *light at the middle is toggled*

2. **Programing**: the program must be written in Java.
   2.1 Ask user for N (#rows and #columns), N must be at least 2.
   2.2 Ask user for initial state of the grid. For convenience, you may ask for a sequence of N x N bits.
   2.3 Ask whether there is a broken light and its position on the grid.
   2.4 Find a way to switch off all lights, if there is one. Display the light states in each move until all lights are OFF. You'll get extra point if the number of moves is minimum.
   2.5 The program should be able to loop for a new game with different input & handle input errors.
   2.6 Your source files (.java) must be in folder Project2_XXX where XXX = ID of the group representative, assuming that this folder is under Maven's "src/main/java" structure. The first lines of all source files must be comments containing English names & IDs of all members.

3. This puzzle is generally known as "light out" puzzle and there are a few variants of it. There are also many approaches to solving this puzzle. But you are required to use graphs and graph algorithms in this project.
   - You can search and use any algorithm or even pieces of code, provided that the sources are properly acknowledged.
   - These sources must be from your own research e.g. paper, textbook, Internet, etc.
   - Using classmates as references is considered cheating.

```
--- exec-maven-plugin:3.0.0:exec (default-cli) @ javasolutions ---
Enter number of rows for 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

Set broken light (Y/N) ?  n

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
```

Demo 1: 2x2 grid, no broken light

```
--- exec-maven-plugin:3.0.0:exec (default-cli) @ javasolutions ---
Enter number of rows for 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

Set broken light (Y/N) ?  y
Enter row of broken light (0-1) = 0
Enter col of broken light (0-1) = 0

3 moves to turn off all lights

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

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

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

Demo 2: 2x2 grid, with broken light

Design a better user interface by yourself. Good user interface doesn't mean fancy output, but rather how your program is easy to understand & to use even without user manual. For example,

- Can user continue playing without having to restart the program?
- Are the instructions clear enough?
- Do you warn about valid input and/or handle invalid input?
- Is the output properly formatted and understandable?

```
--- exec-maven-plugin:3.0.0:exec (default-cli) @ javasolutions ---
Enter number of rows for square grid = 3
Enter initial states (9 bits, left to right, line by line) = 000101010

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) ?  n

6 moves to turn off all lights

>>> Move 1 : turn off 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 off 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 on 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 off 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 off 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 off row 2, 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   |   0
```

Demo 3: 3x3 grid, no broken light

```
--- exec-maven-plugin:3.0.0:exec (default-cli) @ javasolutions ---
Enter number of rows for square grid = 3
Enter initial states (9 bits, left to right, line by line) = 000101010

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) = 2
Enter col of broken light (0-2) = 2

5 moves to turn off all lights

>>> Move 1 : turn on 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 off 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 off 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 off 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 off 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
```

Demo 4: 3x3 grid, with broken light

```
--- exec-maven-plugin:3.0.0:exec (default-cli) @ javasolutions ---
Enter number of rows for square grid = 3
Enter initial states (9 bits, left to right, line by line) = 000101010

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
No solution !!
```

Demo 5: 3x3 grid, with broken light

4. **Report in THAI**: describe at least the following
   4.1 Short user manual
   4.2 Graph data structure
   - What do nodes & edges in graph represent? What conditions do you check in order to add an edge from one node to another?
   - Draw graph with real nodes and edges in Demo 1.
   - Draw graph with real nodes and edges in Demo 2.

   4.3 Other data structures e.g. ArrayList, ArrayDeque, HashSet, HashMap, etc.
   - Explain reason to use each of them.

   4.4 Graph algorithm(s) to switch off all lights. Source code listing without any explanation isn't counted as a proper report.
   - Which graph algorithms (e.g. BFS, DFS, Dijkstra, Kruskal, etc.) are used to solve the puzzle? Explain reason to use each of them. No need to explain how they work.
   - Explain why there is no solution in some cases such as Demo 5. Don't just say that a method returns null or throws an exception, but explain what is wrong with the graph in Demo 5 (compared to Demos 3 & 4) that leads to no solution. Your explanation should be supported by real graph data in these Demos. Show the graphs or write some code to retrieve & compare graph data.

   4.5 Any limitation of your program. Convenient limitation isn't counted as a proper limitation. For example, don't give a limitation that your program will crash if N is negative just because you are too lazy to check for valid input

   4.6 References or declaration of originality
   - If you design the data structures/algorithms and write the whole program by yourself, explicitly declare that everything is done by yourself. But if it is found later that this is not true, it will be counted as cheating
   - Otherwise, valid references/URLs must be given
   - A report without any reference or declaration of originality will get point deduction

Grading

Programming (10 points)

2 points All lights are OFF, for grids with & without broken light. Light states are correct in each move

1 point Minimum number of moves

2 points User interface + exception handling
- This part must be your own effort. Even if 2 groups take algorithms or pieces of code from the same source, it would be impossible to have the same/similar coding for the user interface. Therefore, same (or too similar) user interface coding is considered cheating

5 points Programming (2 points) + OOP with Java Collection and JGraphT 1.5.1 APIs (3 points)
- Although this is not a programming course, your program should still be well structured. Excessive use of static methods as C-style functions is strongly discouraged.
- Even if you use pieces of code from the Internet, try to convert it to proper OOP with Java Collection and JGraphT APIs. Don't implement your own graph data structure or graph algorithms if they are already available in JGraphT.

Report (10 points)

4 points Graph and other data structures, example graphs in Demos 1 & 2

4 points Graph algorithms, comparison between Demos 3-5

2 points Manual, limitation, reference/declaration of originality, others (writing, format, etc.)

Submission

1. A report in only 1 PDF file. The front page must contain names & IDs of everyone in your group
2. Source files (*.java)
3. File readme.txt containing names & IDs of everyone in your group
4. Put all files in folder Project2_XXX (see 2.6) and zip the folder
    - The group representative submits the whole project to Google Classroom
    - The other group members submit only readme.txt to Google Classroom