**The Game of Life**

The Game of Life was invented by the British mathematician John Conway in 1970. It is a zero-player game with no winner or loser, but it is equivalent to a deterministic automaton. The game is played in a matrix of N×M cells, each of which may be either ‘alive’ or ‘dormant’ at each moment, and the state of the cell matrix evolves according to the following rules: A cell in the alive state will enter the dormant state in the next second if there are fewer than two surviving cells around it. A cell in the surviving state will enter the dormant state in the next second due to overcrowding if there are more than three cells in the surviving state around it.

A surviving cell surrounded by two or three surviving cells at a given second will remain surviving for the next second.

Suppose the number of surrounding cells in the surviving state is precisely three at a given second. In that case, a dormant cell is considered more suitable for survival and will enter the surviving state in the next second.

The scope of ‘surrounding’ includes the cells above, below, left, right, top left, bottom left, top right and bottom right of a cell, if they exist.

Given an initial state matrix of 5\*5, explore what the valid state is with the most significant number of repetitions in n steps (n>1) and how many times it has been repeated (excluding the initial state). The initial state and the value n will be set as two initial values in the code, which means you don’t need to implement an input function.

A valid state is one in which at least one cell survives. The state matrix is a two-dimensional, integer-type matrix. Surviving cells are denoted by one, and dormant cells are represented by zero. The final output’s length needs to be six rows. The first row is the maximum number of repetitions. Rows 2 to 6 are a 5\*5 matrix representing the states that have been repeated; no spaces need to be printed between the numbers.

Note 1: If the maximum number of repetitions is zero or one, i.e., the valid state doesn’t exist, or all the valid states don’t repeat, then the output matrix is a zero matrix.

Note 2: You need to consider boundary issues. As an example, for a point with coordinates (0, 0) you need to consider (4, 4), (4, 0), (4, 1), (0, 4), (0, 1), (0, 1), (1, 1) and (1, 0).

Example:

Input:

n = 3

Int[][] state = {{0,0,0,0,0},{0,0,0,0,0},{0,1,1,1,0},{0,0,0,0,0},{0,0,0,0,0}}

Output:

2

00000

00100

00100

00100

00000

For this question, after analysis, we can learn that the problem can be split into two sub-problems. The first sub-problem is how to simulate the Game of Life. The second sub-problem is how to count the repeated states. Based on the above analysis, let’s complete the problem step by step (each completed task will earn some marks).

First, let’s create a class as shown below.

1. **public** **class** GameOfLife {
2. **public** **static** **int**[][] state = {
3. {0, 1, 0, 0, 0},
4. {0, 0, 1, 0, 0},
5. {1, 1, 1, 0, 0},
6. {0, 0, 0, 0, 0},
7. {0, 0, 0, 0, 0}};
8. **public** **static** **int** n = 103;
9. **public** **static** HashMap<String, Integer> count = **new** HashMap<String, Integer>();
10. }

**Problem 1:** **(4 marks)** Implement a function: **public static int returnAlive(int x, int y){}**. Input two integers, x and y. Return the alive cell’s count around the state[x][y]. Example: Input: x=2, y=4, Output: 1

**Problem 2:** **(4 marks)** Implement a function: **public static void goNextState(){}**. When calling this function, the state variable should change to the next state’s value. Example: If we call this function now, the state’s value should change as follows:

1. state = {
2. {0, 0, 0, 0, 0},
3. {0, 1, 0, 0, 0},
4. {0, 1, 0, 0, 0},
5. {0, 1, 0, 0, 0},
6. {0, 0, 0, 0, 0}};

**Problem 3:** **(4 marks)** HashMap is generally used for counting duplicate content. However, a two-dimensional matrix is not suitable for using a key of the HashMap. We provide a solution here that converts this 5\*5 0-1 matrix into a 25-length 0-1 String. Implement a function: **public static String getStateStr(){}.** When we call this function, it should return a 25-length String that represents the state matrix.

Example: If we call this function now, this function should return the following String: 0000000000011100000000000

**Problem 4:** **(2 marks)** Note that we only count the valid state in this problem. Write a function to check if a 25-length String represents a valid state. **public static boolean checkStateStr(String str){}**

Example: checkStateStr(“0000000000000000000000000”), output=false

**Problem 5:** **(4 marks)** We need to print the ans matrix at the end. Write a function to print a 5\*5 matrix based on a 25-length String input. **public static void printState(String stateStr){}**

Example: printState(“0000000000011100000000000”)

Print this matrix as an output:

00000

00000

01110

00000

00000

**Problem 6:** **(12 marks)** With the help of the above function, complete this problem within the main function.