Game of Life

According to the Wikipedia's article: "The **Game of Life**, also known simply as **Life**, is a cellular automaton devised by the British mathematician John Horton Conway in 1970."

Given a *board* with *m* by *n* cells, each cell has an initial state *live* (1) or *dead* (0). Each cell interacts with its eight neighbors (horizontal, vertical, diagonal) using the following four rules (taken from the above Wikipedia article):

1. Any live cell with fewer than two live neighbors dies, as if caused by under-population.
2. Any live cell with two or three live neighbors lives on to the next generation.
3. Any live cell with more than three live neighbors dies, as if by over-population..
4. Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

Write a function to compute the next state (after one update) of the board given its current state.

Follow up:

1. Could you solve it in-place? Remember that the board needs to be updated at the same time: You cannot update some cells first and then use their updated values to update other cells.
2. In this question, we represent the board using a 2D array. In principle, the board is infinite, which would cause problems when the active area encroaches the border of the array. How would you address these problems?

Credits:
Special thanks to @jianchao.li.fighter for adding this problem and creating all test cases.

Solution 1

Since the board has ints but only the 1-bit is used, I use the 2-bit to store the new state. At the end, replace the old state with the new state by shifting all values one bit to the right.

```
void gameOfLife(vector<vector<int>>& board) {
    int m = board.size(), n = m ? board[0].size() : 0;
    for (int i=0; i<m; ++i) {
        for (int j=0; j<n; ++j) {
            int count = 0;
            for (int I=max(i-1, 0); I<min(i+2, m); ++I)
                for (int J=max(j-1, 0); J<min(j+2, n); ++J)
                    count += board[I][J] & 1;
            if (count == 3 || count - board[i][j] == 3)
                board[i][j] |= 2;
        }
    }
    for (int i=0; i<m; ++i)
        for (int j=0; j<n; ++j)
            board[i][j] >>= 1;
}
```

Note that the above `count` counts the live ones among a cell's neighbors and the cell itself. Starting with `int count = -board[i][j]` counts only the live neighbors and allows the neat

```
if ((count | board[i][j]) == 3)
```

test. Thanks to aileenbai for showing that one in the comments.

written by StefanPochmann original link here

Solution 2

To solve it in place, we use 2 bits to store 2 states:

```
[2nd bit, 1st bit] = [next state, current state]

- 00  dead (next) <- dead (current)
- 01  dead (next) <- live (current)
- 10  live (next) <- dead (current)
- 11  live (next) <- live (current)
```

- In the beginning, every cell is either `00` or `01`.
- Notice that `1st` state is independent of `2nd` state.
- Imagine all cells are instantly changing from the `1st` to the `2nd` state, at the same time.
- Let's count # of neighbors from `1st` state and set `2nd` state bit.
- Since every `2nd` state is by default dead, no need to consider transition `01 -> 00`.
- In the end, delete every cell's `1st` state by doing `>> 1`.

For each cell's `1st` bit, check the 8 pixels around itself, and set the cell's `2nd` bit.

- Transition `01 -> 11`: when `board == 1` and `lives >= 2 && lives <= 3`.
- Transition `00 -> 10`: when `board == 0` and `lives == 3`.

To get the current state, simply do

```
board[i][j] & 1
```

To get the next state, simply do

```
board[i][j] >> 1
```

Hope this helps!

```java
public void gameOfLife(int[][] board) {
    if(board == null || board.length == 0) return;
    int m = board.length, n = board[0].length;

    for(int i = 0; i < m; i++) {
        for(int j = 0; j < n; j++) {
            int lives = liveNeighbors(board, m, n, i, j);

            // In the beginning, every 2nd bit is 0;
            // So we only need to care about when the 2nd bit will become 1.
            if(board[i][j] == 1 && lives >= 2 && lives <= 3) {
                board[i][j] = 3; // Make the 2nd bit 1: 01 ---> 11
            }
            if(board[i][j] == 0 && lives == 3) {
                board[i][j] = 2; // Make the 2nd bit 1: 00 ---> 10
            }
        }
    }

    for(int i = 0; i < m; i++) {
        for(int j = 0; j < n; j++) {
            board[i][j] >>= 1;  // Get the 2nd state.
        }
    }
}

public int liveNeighbors(int[][] board, int m, int n, int i, int j) {
    int lives = 0;
    for(int x = Math.max(i - 1, 0); x <= Math.min(i + 1, m - 1); x++) {
        for(int y = Math.max(j - 1, 0); y <= Math.min(j + 1, n - 1); y++) {
            lives += board[x][y] & 1;
        }
    }
    lives -= board[i][j] & 1;
    return lives;
}
```

written by yavinci original link here

Solution 3

```
public class Solution {
int[][] dir ={{1,-1},{1,0},{1,1},{0,-1},{0,1},{-1,-1},{-1,0},{-1,1}};
public void gameOfLife(int[][] board) {
    for(int i=0;i<board.length;i++){
        for(int j=0;j<board[0].length;j++){
            int live=0;
            for(int[] d:dir){
                if(d[0]+i<0 || d[0]+i>=board.length || d[1]+j<0 || d[1]+j>=board[
0].length) continue;
                if(board[d[0]+i][d[1]+j]==1 || board[d[0]+i][d[1]+j]==2) live++;
            }
            if(board[i][j]==0 && live==3) board[i][j]=3;
            if(board[i][j]==1 && (live<2 || live>3)) board[i][j]=2;
        }
    }
    for(int i=0;i<board.length;i++){
        for(int j=0;j<board[0].length;j++){
            board[i][j] %=2;
        }
    }
}
```

}

written by DREAM123 original link here