1. Game of Life

According to the [Wikipedia’s article](https://en.wikipedia.org/wiki/Conway's_Game_of_Life): “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](https://en.wikipedia.org/wiki/Moore_neighborhood) (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. The next state is created by applying the above rules simultaneously to every cell in the current state, where births and deaths occur simultaneously.

**Example:**

Input:   
[  
 [0,1,0],  
 [0,0,1],  
 [1,1,1],  
 [0,0,0]  
]  
Output:   
[  
 [0,0,0],  
 [1,0,1],  
 [0,1,1],  
 [0,1,0]  
]

**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?

**解法1** 将原矩阵复制下来，按照游戏规则修改原来的矩阵

class Solution {  
public:  
 void gameOfLife(vector<vector<int>>& board) {  
 vector<vector<int>>tmp\_board(board.begin(), board.end());  
 int m = board.size(), n = board[0].size();  
 for(int i = 0; i < m; ++i){  
 for(int j = 0; j < n; ++j){  
 int cnt = 0;  
 for(int k = 0; k < 8; ++k){  
 int tmp\_x = i + dx[k], tmp\_y = j + dy[k];  
 if(valid(tmp\_x, tmp\_y, m, n) && tmp\_board[tmp\_x][tmp\_y]){  
 cnt++;  
 }  
 }  
 if(tmp\_board[i][j] == 1){  
 if(cnt < 2 || cnt > 3){  
 board[i][j] = 0;  
 }else{  
 board[i][j] = 1;  
 }  
 }else{  
 if(cnt == 3)board[i][j] = 1;  
 else board[i][j] = 0;  
 }  
 }  
 }  
 }  
private:  
 int dx[8] = {-1, 0, -1, -1, 0, 1, 1, 1};  
 int dy[8] = {0, -1, -1, 1, 1, 0, -1, 1};  
 bool valid(int x, int y, int m, int n){  
 if(x < 0 || x >= m || y < 0 || y >= n)return false;  
 return true;  
 }  
};

**解法2** 原地修改，空间复杂度。使用多个状态：

* 0：原来是0，新的还是0
* 1：原来是1，新的还是1
* 2：原来是0，新的是1
* 3：原来是1，新的是0

按照行顺序更新时，对于每个cell，左、上、左上、右上是被更新了，剩下四个没有更新，按照对应的数值统计出在原始矩阵中的数字，然后更新当前cell，最后遍历一遍，把2和3分别修改成1和0

class Solution {  
public:  
 void gameOfLife(vector<vector<int>>& board) {  
 int m = board.size(), n = board[0].size();  
 for(int i = 0; i < m; ++i){  
 for(int j = 0; j < n; ++j){  
 int cnt = 0;  
 for(int k = 0; k < 4; ++k){  
 int tmp\_x = i + dx[k], tmp\_y = j + dy[k];  
 if(valid(tmp\_x, tmp\_y, m, n) &&  
 (board[tmp\_x][tmp\_y] == 3 || board[tmp\_x][tmp\_y] == 1)){  
 cnt++;  
 }  
 }  
 for(int k = 4; k < 8; ++k){  
 int tmp\_x = i + dx[k], tmp\_y = j + dy[k];  
 if(valid(tmp\_x, tmp\_y, m, n) && board[tmp\_x][tmp\_y] == 1){  
 cnt++;  
 }  
 }  
 if(board[i][j] == 1){  
 if(cnt < 2 || cnt > 3){  
 board[i][j] = 3;  
 }else{  
 board[i][j] = 1;  
 }  
 }else{  
 if(cnt == 3)board[i][j] = 2;  
 else board[i][j] = 0;  
 }  
   
 }  
 }  
 for(int i = 0; i < m; ++i){  
 for(int j = 0; j < n; ++j){  
 if(board[i][j] == 2)board[i][j] = 1;  
 else if(board[i][j] == 3)board[i][j] = 0;  
 }  
 }  
 }  
private:  
 int dx[8] = {-1, 0, -1, -1, 0, 1, 1, 1};  
 int dy[8] = {0, -1, -1, 1, 1, 0, -1, 1};  
 bool valid(int x, int y, int m, int n){  
 if(x < 0 || x >= m || y < 0 || y >= n)return false;  
 return true;  
 }  
};