Problem 529: Minesweeper

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

Let's play the minesweeper game (

Wikipedia

,

online game

)!

You are given an

m x n

char matrix

board

representing the game board where:

'M'

represents an unrevealed mine,

'E'

represents an unrevealed empty square,

'B'

represents a revealed blank square that has no adjacent mines (i.e., above, below, left, right, and all 4 diagonals),

digit (

'1'

to

'8'

) represents how many mines are adjacent to this revealed square, and

'X'

represents a revealed mine.

You are also given an integer array

click

where

click = [click

r

, click

c

]

represents the next click position among all the unrevealed squares (

'M'

or

'E'

).

Return

the board after revealing this position according to the following rules

:

If a mine

'M'

is revealed, then the game is over. You should change it to

'X'

.

If an empty square

'E'

with no adjacent mines is revealed, then change it to a revealed blank

'B'

and all of its adjacent unrevealed squares should be revealed recursively.

If an empty square

'E'

with at least one adjacent mine is revealed, then change it to a digit (

'1'

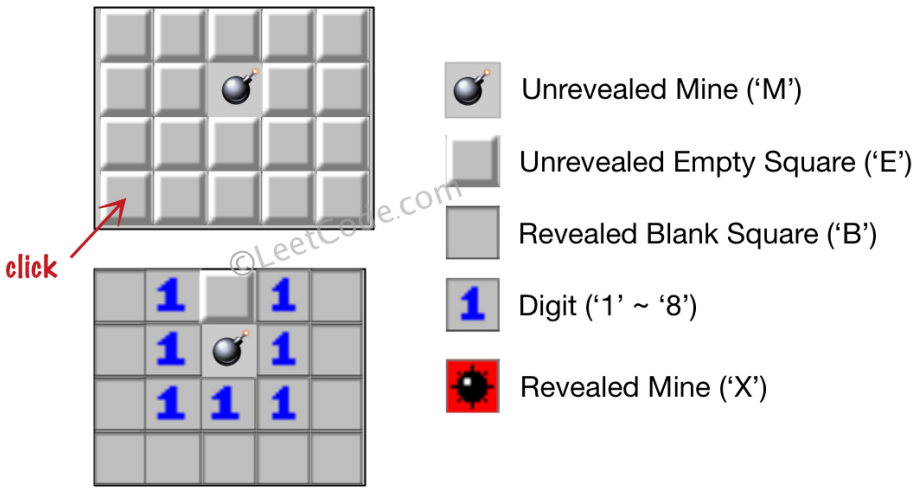
to

'8'

) representing the number of adjacent mines.

Return the board when no more squares will be revealed.

Example 1:



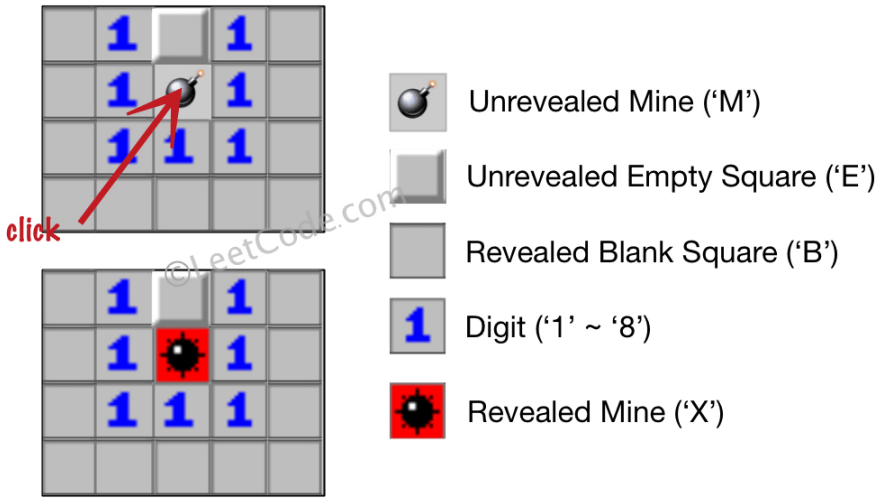
Input:

board = [["E","E","E","E","E"],["E","E","M","E","E"],["E","E","E","E","E"],["E","E","E","E","E"]],
click = [3,0]

Output:

[["B","1","E","1","B"],["B","1","M","1","B"],["B","1","1","1","B"],["B","B","B","B","B"]]

Example 2:



Input:

board = [["B","1","E","1","B"],["B","1","M","1","B"],["B","1","1","1","B"],["B","B","B","B","B"]], click = [1,2]

Output:

[["B","1","E","1","B"],["B","1","X","1","B"],["B","1","1","1","B"],["B","B","B","B","B"]]

Constraints:

m == board.length

n == board[i].length

1 <= m, n <= 50

board[i][j]

is either

'M'

,

'E'

,

'B'

, or a digit from

'1'

to

'8'

.

click.length == 2

0 <= click

r

< m

0 <= click

c

< n

board[click

r

][click

c

]

is either

'M'

or

'E'

.

Code Snippets

C++:

```
class Solution {
public:
vector<vector<char>> updateBoard(vector<vector<char>>& board, vector<int>&
click) {

}
};
```

Java:

```
class Solution {
public char[][] updateBoard(char[][] board, int[] click) {

}
}
```

Python3:

```
class Solution:
def updateBoard(self, board: List[List[str]], click: List[int]) ->
List[List[str]]:
```

Python:

```
class Solution(object):
def updateBoard(self, board, click):
"""
:type board: List[List[str]]
```

```
:type click: List[int]
:rtype: List[List[str]]
"""
```

JavaScript:

```
/**
* @param {character[][]} board
* @param {number[]} click
* @return {character[][]}
*/
var updateBoard = function(board, click) {

};
```

TypeScript:

```
function updateBoard(board: string[][], click: number[]): string[][] {

};
```

C#:

```
public class Solution {
public char[][] UpdateBoard(char[][] board, int[] click) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
char** updateBoard(char** board, int boardSize, int* boardColSize, int*
click, int clickSize, int* returnSize, int** returnColumnSizes) {

}
```

Go:

```
func updateBoard(board [][]byte, click []int) [][]byte {

}
```

Kotlin:

```
class Solution {
fun updateBoard(board: Array<CharArray>, click: IntArray): Array<CharArray> {

}
}
```

Swift:

```
class Solution {
func updateBoard(_ board: [[Character]], _ click: [Int]) -> [[Character]] {

}
}
```

Rust:

```
impl Solution {
pub fn update_board(board: Vec<Vec<char>>, click: Vec<i32>) -> Vec<Vec<char>>
{

}
}
```

Ruby:

```
# @param {Character[][]} board
# @param {Integer[]} click
# @return {Character[][]}
def update_board(board, click)

end
```

PHP:

```
class Solution {

/**
* @param String[][] $board
* @param Integer[] $click
* @return String[][]
*/
function updateBoard($board, $click) {

}
}
```

Dart:

```
class Solution {
List<List<String>> updateBoard(List<List<String>> board, List<int> click) {

}
}
```

Scala:

```
object Solution {
def updateBoard(board: Array[Array[Char]], click: Array[Int]):
Array[Array[Char]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec update_board(board :: [[char]], click :: [integer]) :: [[char]]
def update_board(board, click) do

end
end
```

Erlang:

```
-spec update_board(Board :: [[char()]], Click :: [integer()]) -> [[char()]].
update_board(Board, Click) ->
.
```

Racket:

```
(define/contract (update-board board click)
(-> (listof (listof char?)) (listof exact-integer?) (listof (listof char?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minesweeper
* Difficulty: Medium
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
vector<vector<char>> updateBoard(vector<vector<char>>& board, vector<int>&
click) {

}
};
```

Java Solution:

```
/**
* Problem: Minesweeper
* Difficulty: Medium
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
```

```
public char[][] updateBoard(char[][] board, int[] click) {

}
}
```

Python3 Solution:

```
"""
Problem: Minesweeper
Difficulty: Medium
Tags: array, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def updateBoard(self, board: List[List[str]], click: List[int]) ->
List[List[str]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def updateBoard(self, board, click):
"""
:type board: List[List[str]]
:type click: List[int]
:rtype: List[List[str]]
"""
```

JavaScript Solution:

```
/**
* Problem: Minesweeper
* Difficulty: Medium
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {character[][]} board
* @param {number[]} click
* @return {character[][]}
*/
var updateBoard = function(board, click) {

};
```

TypeScript Solution:

```
/**
* Problem: Minesweeper
* Difficulty: Medium
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function updateBoard(board: string[][], click: number[]): string[][] {

};
```

C# Solution:

```
/*
* Problem: Minesweeper
* Difficulty: Medium
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
public class Solution {
public char[][] UpdateBoard(char[][] board, int[] click) {

}
}
```

C Solution:

```
/*
* Problem: Minesweeper
* Difficulty: Medium
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
char** updateBoard(char** board, int boardSize, int* boardColSize, int*
click, int clickSize, int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Minesweeper
// Difficulty: Medium
// Tags: array, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func updateBoard(board [][]byte, click []int) [][]byte {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun updateBoard(board: Array<CharArray>, click: IntArray): Array<CharArray> {

}
}
```

Swift Solution:

```
class Solution {
func updateBoard(_ board: [[Character]], _ click: [Int]) -> [[Character]] {

}
}
```

Rust Solution:

```
// Problem: Minesweeper
// Difficulty: Medium
// Tags: array, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn update_board(board: Vec<Vec<char>>, click: Vec<i32>) -> Vec<Vec<char>>
{

}
}
```

Ruby Solution:

```
# @param {Character[][]} board
# @param {Integer[]} click
# @return {Character[][]}
def update_board(board, click)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param String[][] $board
* @param Integer[] $click
* @return String[][]
*/
function updateBoard($board, $click) {

}
}
```

Dart Solution:

```
class Solution {
List<List<String>> updateBoard(List<List<String>> board, List<int> click) {

}
}
```

Scala Solution:

```
object Solution {
def updateBoard(board: Array[Array[Char]], click: Array[Int]):
Array[Array[Char]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec update_board(board :: [[char]], click :: [integer]) :: [[char]]
def update_board(board, click) do

end
```

```
end
```

Erlang Solution:

```
-spec update_board(Board :: [[char()]], Click :: [integer()]) -> [[char()]].
update_board(Board, Click) ->
.
```

Racket Solution:

```
(define/contract (update-board board click)
(-> (listof (listof char?)) (listof exact-integer?) (listof (listof char?)))
)
```