Problem Name

Cut Off Trees for Golf Event

Description

You are asked to cut off trees in a forest for a golf event. The forest is represented as a non-negative 2D map, in this map:

1. `0` represents the `obstacle` can't be reached.
2. `1` represents the `ground` can be walked through.
3. `The place with number bigger than 1` represents a `tree` can be walked through, and this positive number represents the tree's height.

You are asked to cut off **all** the trees in this forest in the order of tree's height - always cut off the tree with lowest height first. And after cutting, the original place has the tree will become a grass (value 1).

You will start from the point (0, 0) and you should output the minimum steps **you need to walk** to cut off all the trees. If you can't cut off all the trees, output -1 in that situation.

You are guaranteed that no two `trees` have the same height and there is at least one tree needs to be cut off.

```
1 2 3
0 0 4
7 6 5
```

Output: 6

```
1 2 3
0 0 0
7 6 5
```

Output: -1

```
2 3 4
0 0 5
8 7 6
```

Output: 6

IOCE

Input:

An mxn matrix containing integers

Output:

The minimum number of steps you need to walk to cut all of the trees, if you can't cut all of the trees output -1

Constraints:

None

Edge cases:

-- no trees
-- array dimensions are 0x0

Approach pseudo-code

(The approaches listed below are only two of many ways an interviewee could solve this problem. If your candidate uses a different approach it could also be correct. Use these examples to develop a better understanding of the problem so that you can give proper feedback and support to your interviewee)

```
// PSEUDOCODE
// handle edge case of empty array
// Iterate through matrix and push tree location and value to queue
// Sort array based on integer value
// Define variables start && totalSteps
// While we still have trees left in our queue
     // dequeue tree location and value and set as target tree
     // count minimum steps from start to target tree
     // if we can't get to target tree
          // return -1
     // Increment totalSteps by steps minimum steps to target tree
     // Change start location to target tree location
// return totalSteps
```

```
// PSEUDOCODE FOR COUNTING MINIMUM STEPS TO TARGET TREE (BFS
APPROACH)
// A* Search and Hadlock's Algorithm would also work here
// Define the 4 directions we can go
// define stepCount at 0
// define visited as an empty array and populate with grid the same
size as the input grid setting each value to false
// define an empty array to act as a queue
// push our starting location into the queue
// set starting location in visited array to true
// while queue.length is greater than 0
     // iterate over queue
          // set current to first position in queue
          // if current === target tree
               // return stepCount
          // for each possible direction
               // if new position is inside grid, has not already
been visited, and is not impassable (has the value of 0)
                    // push new position into queue
                    // set visited[newPosition] to true
     // increment stepCount
// return -1
```

Hints (sometimes)

Note: Hints won't always be provided and depending on how far your interviewee got before getting stuck, won't always be helpful. The interviewer should choose hints from this list at their discretion. REMEMBER, the goal is to nudge them out of being stuck, not show them the answer.

- How can we find the minimum steps to get to a new location in a grid?
- Can an additional data structure help us here?
- Are we properly exitting out of our algorithm when we find out there is no solution?

Code

```
const minStep = (forset, start, tree, rows, cols) => {
  let directions = [[0, 1], [0, -1], [1, 0], [-1, 0]];
  let step = 0;
  let visited = [];
  for (let i = 0; i < rows; i++) {
    visited.push([]);
    for (let j = 0; j < cols; j++) {
      visited[i][j] = false;
    }
  }
  let queue = [];
  queue.push(start);
  visited[start[0]][start[1]] = true;
  while (queue.length > 0) {
    for (let i = 0; i < queue.length; i++) {
      let current = queue.shift();
      if (current[0] === tree[0] && current[1] === tree[1]) {
        return step;
      }
      for (let c = 0; c < directions.length; c++) {
        let direction = directions[c];
        let newRow = current[0] + direction[0];
        let newCol = current[1] + direction[1];
        if (newRow > -1 && newRow < rows && newCol > -1 || newCol <
cols || !visited[newRow][newCol] || forest[newRow][newCol] !== 0) {
          queue.push([nr, nc]);
          visited[nr][nc] = true;
        }
      }
    }
    step++;
  }
  return -1;
}
const cutOffTree = forest => {
  if (forest === null || forest.length === 0) return 0;
```

```
  let rows = forest.length;
  let cols = forest[0].length;
  let queue = [];
  for (let i = 0; i < rows; i++) {
    for (let j = 0; j < cols; j++) {
      if (forest[i][j] > 1) {
        queue.push([i, j, forest[i][j]]);
      }
    }
  }
  queue.sort((a, b) => a[2] - b[2]);
  let start = [0, 0];
  let totalSteps = 0;
  while (queue.length > 0) {
    let tree = queue.shift();
    let step = minStep(forest, start, tree, rows, cols);
    if (step < 0) {
      return -1;
    }
    totalSteps += step;
    start[0] = tree[0];
    start[1] = tree[1];
  }
  return totalSteps;
};
```