Recursion: Counting adjacent filled cells in a grid

You have a two-dimensional grid of cells, each of which may be filled or empty. Filled cells which are connected form what is called a *blob* (for lack of a better word). There may be several blobs on a grid. The problem is to write a recursive function that returns the size of the blob containing a specified cell.

For example, for the grid below, countBlob at row=0, col=3 is 3.

```
    * *
    *
* *
* * *   *
* *     *
```

How would you write a function to solve this problem? Essentially you need to check the current cell, its neighbors, the neighbors of its neighbors, and so on.

We'll try to think of this problem recursively. Here are some facts that help build the intuition.

Base case.

When you try to write a recursive method, **always** start from the base cases. What are the base cases for counting the blob?

For any cell at location (row, col) there are 3 possibilities:

—(row,col) may be out of bounds

—the cell (row,col) may be empty

—the cell (row, col) may be filled

In the first two cases the size of the blob containing the cell (row, col) is zero because there is no blob. For the last case we need to go into recursion.

Defining the recursion.

We must define the size of the blob containing the filled cell (row, col) in terms of the size of one or more smaller blobs. If we *mark* the filled cell (row, col) when we visit it then we can redefine the problem as follows:

The size of the blob containing the filled cell (row, col) is 1 + the sizes of any blobs touching the now *marked* cell (row, col).

What does it mean that a blob touches the cell? A blob touches the cell if and only if it contains a neighbour cell of cell. Thus we can further clarify our redefinition as:

The size of the blob containing the filled cell (row, col) is 1 + the sizes of any blobs containing a neighbour cell of the *marked* cell (row, col).

We have redefined the problem in terms of smaller problems of the same type. There are eight neighbour cells of the cell (row, col) and each one must be visited.

The algorithm.

Here is what we have so far¿

```
if (the cell is outside the grid) then return  0

if (the cell is EMPTY or MARKED) then return 0

//else
mark the cell
return  1 + the counts of the cell's eight neighbours.
```

Some notes.

countBlob() calls itself eight times, each time a different neighbour of the current cell is visited. The cells are visited in a clockwise manner starting with the neighbour above and to the left.

As the problem size diminishes will you reach the base cases?

Everytime the routine visits a filled cell it marks it BEFORE it visits its neighbours, reducing the size of the blob by one. Eventually all of the filled cells in the blob will be marked and the routine will encounter nothing but base cases.

If a cell would not be *marked* before the recursive calls, then the cell would be counted more than once since it is a neighbour of each of its eight neighbours. In fact a much worse problem would occur. When each neighbour of the cell is visited, countBlob() is called again on the current cell. Thus if the cell were still FILLED an infinite sequence of calls would be generated.

A side effect of countBlob() is that the blob containing the cell is marked. In other words, erased. We need to restore copy of the grid before another call to countBlob().