Grid-Cell Neighborhoods

Assumptions

1. You have a two-dimensional array in which every element (cell) contains a single, signed number.
    a. Assume the height of the array (the number of rows) is *H*, and that *H* > 0.
    b. Assume the width of the array (the number of columns) is *W*, and that *W* > 0.
    c. For this exercise, it does not matter what sort of number the cells contain; it's only important which values are positive or not.
    d. Assume cell locations are written here as (Y, X) or (row, column).
2. You have a distance threshold, *N*, and you can assume *N* ≥ 0.
3. Assume you are using a Manhattan distance function. From (2,1) to (0,4) would be the sum of the differences in the two dimensions: |2-0| + |1-4| = 2 + 3 = 5
4. For the purposes of the initial exercise, the array dimensions do not wrap. That is, the first column should not be considered adjacent to the last column (unless *W* < 3), and the top row should not be considered adjacent to the bottom row (unless *H* < 3).

The task

The task is to write a routine that will return the number of cells that fall within *N* steps of any positive values in the array. Note:

1. Each cell in the neighborhood should be counted only once, no matter how many positive values it is near. (There is a specific example of this case later.)
2. The cell containing the positive value should also count (as a 0-distance member) as part of its own neighborhood.
3. You may use any language you like, but it should be able to run. That is, please do not merely provide a pseudo-code sketch of how you would solve the problem.

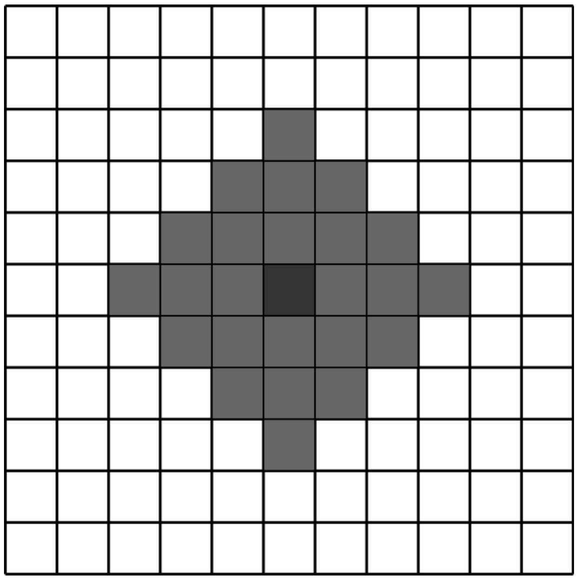
Examples

Each of these examples is named and includes both an illustration of the problem-solved and the expected return value from your routine.

NB: Even though all of these examples use an 11x11 array, that is only to simplify the visual presentation; that size is NOT a constant. Your solution should work on arrays of any valid size as described in *Assumptions*, above.

Example 1: One positive cell fully contained; *N*=3

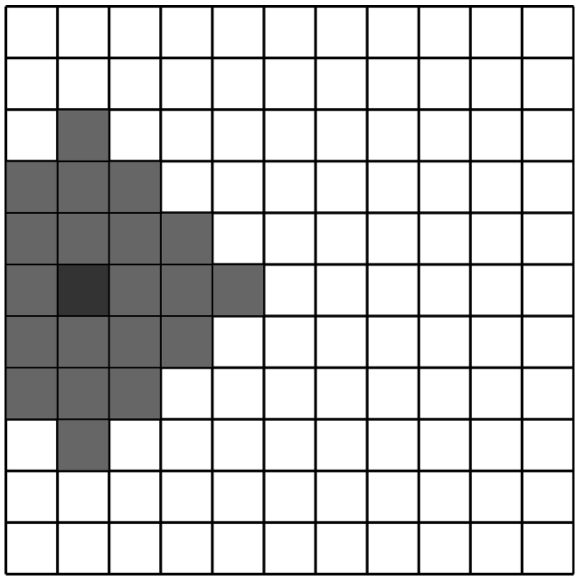
In this example, there is only one positive value, and it is located near the center of a grid that is large enough to contain the *N*=3 neighborhood comfortably.



There are 25 cells in the *N*=3 neighborhood of this single, positive value (including the cell containing the positive value itself).

Example 2: One positive cell near an edge; *N*=3

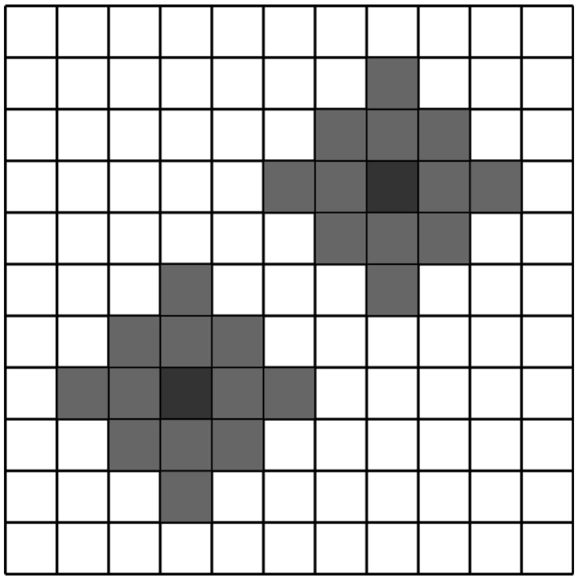
This is similar to *Example 1*, except that the positive cell is less than 3 steps from the left-hand side of the array.



There are only 21 cells in this *N*=3 neighborhood, because the array edges do not wrap. Four of the cells from *Example 1* have "fallen off" the left-hand edge of the array.

Example 3: Two positive values with disjoint neighborhoods; *N*=2

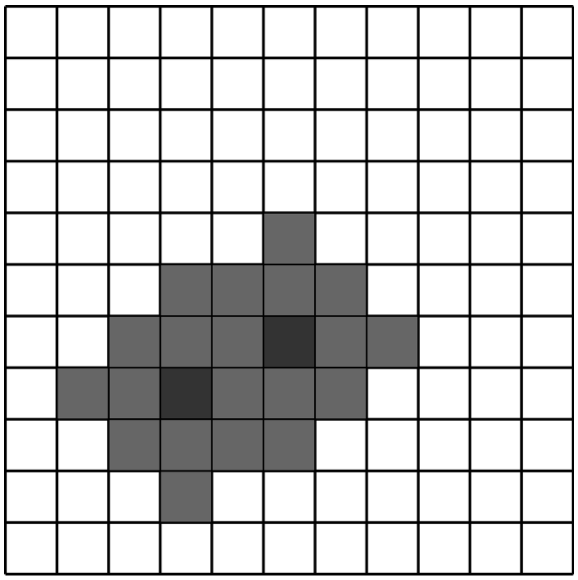
Note that, unlike the previous examples, we are using a different neighborhood radius. Whereas we had been using *N*=3, this example is now using only *N*=2. Your code should work with any valid distance threshold.



Each of the positive values has a 13-cell neighborhood, so the total number of cells within *N*=2 steps of a positive value is 26.

Example 4: Two positive values with overlapping neighborhoods; *N*=2

This example shows that it is perfectly reasonable for the neighborhoods around positive values to overlap.



There are 22 cells that are within N=2 steps of either of the positive cell values in this array. (As an kind of supplemental unit test for your understanding, which are the four cells that are members of both neighborhoods independently?)

Warnings

These examples should not be considered exhaustive. They are only provided to clarify the task your code is to perform. A prudent coder might consider the following cases beyond what has been presented here:

1. multiple positive values whose neighborhoods not only overlap, but that "run off" one or more edges of the array
2. multiple positive values that are directly adjacent
3. one or more positive values that are located in a corner
4. oddly shaped arrays: 1x21; 1x1; 10x1; 2x2; etc.
5. cases where $N \gg \max(W, H)$; even if your routine returns the correct result, what happens to its runtime?
6. case where $N = 0$