Living on a Random Torus

Saad Mneimneh

We don't see doughnut shapes when we look up in the sky. But what if the Earth we live on were shaped like a torus instead of a sphere? This remote, but otherwise intriguing possibility, lead me to explore the formation of islands and pools in a toroidal world. In this exposition, I start with a programming perspective on counting islands and pools, and finish by presenting a probabilistic analysis of a random torus with a ratio p/q of land to water ($p+q=1$).

People on Earth have always been fascinated by the golden ratio $\phi = \frac{1+\sqrt{5}}{2}$. I presume the same would be true for the inhabitants of a torus. As it turns out, in counting islands and pools, the *divine* law of the skies

$$\frac{1}{p} = \left\{\frac{p+q}{p} = \frac{p}{q}\right\} = \phi$$

exhibits itself on a random torus.

1 The Birth of a Torus

It takes a spark of imagination to consider the possibility of living on a planet in the shape of a torus (a doughnut). But when teaching introductory programming to students who have just learned about two-dimensional arrays, all you need is the modulo operator (defined below for $x \in \mathbb{Z}$ and $n \in \mathbb{N}$).

$$x \bmod n = x - n\left\lfloor \frac{x}{n} \right\rfloor$$

where $\lfloor x/n \rfloor$ is the largest integer $\leq x/n$. The modulo operator is widely recognized as the remainder (an integer) in the division of x by n, ranging from 0 to $n-1$. Table 1 shows how $x \bmod n$ maps $\mathbb{Z}$ to the set $\{0, 1, 2, \ldots, n-1\}$.

If we denote the entry at the i^{th} row and the j^{th} column of an $m \times n$ array by $a[i,j]$, where $0 \leq i \leq m-1$ and $0 \leq j \leq n-1$, then $a[i, j-1]$, $a[i, j+1]$, $a[i-1, j]$, and $a[i+1, j]$ represent the *neighbors* of $a[i,j]$. However, the average

x	$\ldots$	-1	0	1	2	$\ldots$	$n-1$	n	$\ldots$
$x \bmod n$	$\ldots$	$n-1$	0	1	2	$\ldots$	$n-1$	0	$\ldots$

Table 1: The modulo operator wraps around as it cycles through the integers from 0 to $n-1$.

programmer can immediately tell that a handful of checks are needed before making access to these neighbors, since some of them are non-existent when $a[i, j]$ lies on the array's boundary. This complicates almost every task one could imagine performed on the array; conditional statements must be inserted everywhere.

In such a programming nightmare, the modulo operator can be a blessing: Given an $m \times n$ array, the neighbors of $a[i, j]$ can be safely defined as in Figure 1.

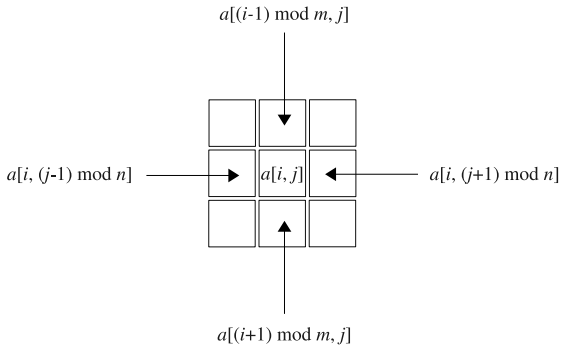

Figure 1: Every $a[i, j]$ has all the neighbors.

The above definitions eliminate boundaries and identify row m with row 0, and column n with column 0. The end result is the folding of the two-dimensional array into a torus, as shown in Figure 2.

The torus is born, and all that remains to be done is adding some life to it! Hence, for a simple programming exercise, let us set every $a[i, j]$ to either a 1 (land) or a 0 (water), and count the islands and pools that form on the torus.

2 Islands and Pools

Following the ideas in the previous section, our torus is given by an $m \times n$ array in which the entry $a[i, j]$ represents either land ($a[i, j] = 1$) or water ($a[i, j] = 0$), and where neighbors of $a[i, j]$ are as defined in Figure 1.

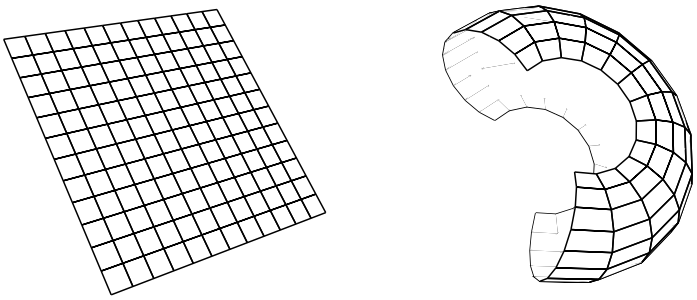

Figure 2: Folding a two-dimensional array into a torus. On the left, the array, and on the right, the folding in action showing a partially folded torus.

An island is intuitively understood as a set of neighboring lands (entries with $a[i, j] = 1$), but a precise definition will depend on how we expect to walk on the torus; it will also have implications on the definition of pools, since islands must be separated by water. A walk on the torus consists of moving through neighbors, on land, and without crossing any waters. Moreover, islands cannot overlap, leading to the following definition.

Definition 1 (Island) *An island is a maximal set of 1s that are reachable from one another by moving through neighbors and without crossing any 0s.*

A note is in order here. I have always been an advocate of infusing algorithmic and mathematical aspects when teaching about programming. The term "maximal" in the above definition most of the times awakens the attention of the students, as it sounds like "maximum" but is not quite the same word. After explaining what maximal means (not a proper subset of some other set), it is often interesting to ponder on the equivalence of the two flavors: the programmer's perspective (more pragmatic) in which islands do not overlap, and the mathematician's perspective (more of a definition) in which islands are maximal. Oddly enough, it is the second one that gives more insight when it comes to writing a program for counting islands.

Similarly, neighboring waters must be part of the same pool, and pools cannot overlap. In addition, the definition of an island implies that the diagonally situated $a[i, j] = 0$ and $a[(i \pm 1) \bmod m, (j \pm 1) \bmod n] = 0$ must be in the same pool to justify the inability to cross diagonally from one land to another.

Definition 2 (Pool) *A pool is a maximal set of 0s that are reachable from one another by moving through neighbors or diagonally, and without crossing any 1s.*

It is this asymmetry in the formation of islands and pools that will create the interesting behavior discussed in the following sections. Figure 3 shows an example of this formation on a random torus.

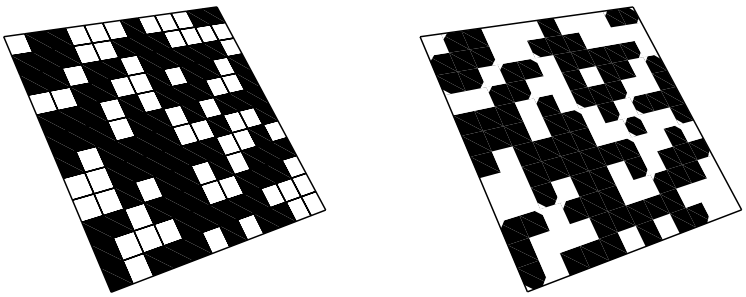

Figure 3: From a two-dimensional array to islands and pools on the torus, showing a total of four islands (black) and two pools (white). Avoid the overcounting of islands and pools when neighbors wrap around in modulo m and/or n.

3 Let's Count

In the Appendix, I present a pseudocode for counting islands and pools on a torus. The code is self explanatory. Upon seeing a land (water) for the first time, it is marked as visited, the rest of the island (pool) is recursively visited by moving through neighbors (and diagonally). When every branch of the recursion stops, the island (pool) is maximal, and a count is incremented. Perhaps it is now a good juncture for examining the pseudocode to appreciate how the introduction of the modulo operator freed us from a bundle of conditional statements (not to mention the additional interesting effect of creating the torus). This should help to focus on aspects of the program that are more important than messing with if-then-else.

In order to count islands and pools, I set $a[i, j] = 1$ with probability p and $a[i, j] = 0$ with probability $q = 1 - p$, independently for each entry. I performed 100 runs of the program to obtain an average number of islands and an average number of pools. I scaled the averages by the torus size mn . I used $m = n = 100$, and varied p from 0 to 1. The results are shown in Figure 4.

Looking at Figure 4, some of the "fun" questions that could accompany this programming exercise are (assuming mn is large enough):

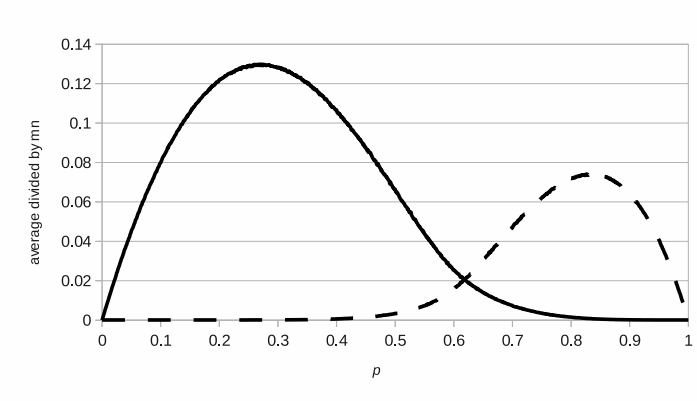

Figure 4: Average number of islands (solid) and pools (dashed) divided by the torus size mn as a function of p.

- For a given p, what is the average number of islands divided by the torus size?
- What happens in the special case of $m = 1$ (or $n = 1$)?
- For what value of p do we get equal averages for islands and pools?

The first question is simply a means to running the program and getting some output. The second question is to provoke some thinking about a special case, which is easier to analyze and might be important for the understanding of the general case. The third question is an invitation to explore further properties and, for the ambitious, perhaps figure out that *special* number between 0.6 and 0.7 in Figure 4.

4 Analysis of a Random Torus

I will perform a probabilistic analysis to show that the expected number of islands and the expected number of pools satisfy:

$$\lim_{m,n\to\infty} \frac{E[\#islands] - E[\#pools]}{mn} = pq(q - p^2)$$

where $\#islands$ is the number of islands, $\#pools$ is the number of pools, and $E[\ ]$ is the standard notation for expectation. This in turn will reveal an interesting property of a random torus that can be linked to the golden ratio (as one might suspect already by looking at Figure 4). It will also provide estimates for $E[\#islands]$ when $E[\#islands] \gg E[\#pools]$, and $E[\#pools]$ when $E[\#islands] \ll E[\#pools]$.

4.1 A Powerful Tool: The Linearity of Expectation

In probability theory, the *expectation* of a random variable X is defined as (somewhat an "average"):

$$E[X] = \sum_x xP(X = x)$$

when the sum exists, where the sum is over all values x that X takes on. The expectation of the sum of two (or more) random variables is the sum of their expectations. For instance,

$$E[X + Y] = E[X] + E[Y]$$

This property is called the *linearity of expectation*, which holds regardless of whether X and Y are independent or not. It is notable here, however, that independence is sufficient for $E[XY] = E[X]E[Y]$.

The linearity of expectation is the key property that enables us to perform probabilistic analysis using *indicator random variables*. An indicator random variable is defined as

$$Z = \begin{cases} 1 & \text{if some event occurs} \\ 0 & \text{otherise} \end{cases}$$

In particular, if Z is an indicator random variable, then

$$E[Z] = 1 \cdot P(Z = 1) + 0 \cdot P(Z = 0) = P(Z = 1)$$

Therefore, if $X = \sum_i X_i$ is the sum of indicator random variables, then

$$E[X] = E\Big[\sum_i X_i\Big] \underbrace{=}_{\substack{\text{linearity} \\ \text{of expectation}}} \sum_i E[X_i] \underbrace{=}_{\substack{\text{indicator} \\ \text{random variable}}} \sum_i P(X_i = 1)$$

To illustrate the power of this technique, consider n independent tosses of a coin with probability p for heads and probability $q = 1 - p$ for tails, and let X be the total number of heads. The expectation of X is given by:

$$E[X] = \sum_{k=0}^{n} kP(X = k) = \sum_{k=0}^{n} k \left(\begin{array}{c} n \\ k \end{array} \right) p^k q^{n-k}$$

To calculate the above sum might be cumbersome. Instead, consider an indicator random variable X_i, such that $X_i = 1$ if the i^{th} toss reveals a head, and $X_i = 0$ otherwise. Then $X = \sum_{i=1}^{n} X_i$. So $E[X] = \sum_{i=1}^{n} P(X_i = 1) = np$.

4.2 Pólya's Advice and a Special Case

If $\#islands$ is the number of islands, then

$$E[\#islands] = \sum_k kP(\#islands = k)$$

It is not entirely obvious how to capture the event of having k islands. Therefore, I will follow George Pólya's advice:

"If you can't solve a problem,
there is an easier problem you can solve.
find it."

And here it is: Consider a special case of the problem in which $m = 1$, so the torus becomes a $1 \times n$ torus, i.e. a circular (one-dimensional) array, where $n > 1$. Again,

$$E[\#islands] = \sum_{k=0}^{\lfloor n/2 \rfloor} kP(\#islands = k)$$

is not an obvious quantity. This time, however, it is easy to capture the islands using indicator random variables. Let X_j be an indicator random variable for the event $a[0, j] = 1 \wedge a[0, j + 1] = 0$ (all indices are taken modulo n), and call $a[0, j]$ in such an event the *edge* of its island. Let Y be an indicator random variable for the event $a[0, 0] = a[0, 1] = \ldots = a[0, n - 1] = 1$. Then

$$\#islands = \sum_{j=0}^{n-1} X_j + Y$$

In other words, the number of islands is the number of times we transition from 1 to 0 when moving from $a[0, j]$ to $a[0, j + 1]$ (number of edges), or simply 1 if $a[0, j] = 1$ and no such transitions occur (no edges). Observe that the X's and Y are highly dependent, but this is irrelevant for the linearity of expectation.

$$E[\#islands] = \sum_{j=0}^{n-1} P(X_j = 1) + P(Y = 1) = npq + p^n$$

and by symmetry,

$$E[\#pools] = nqp + q^n$$

Therefore,

$$\lim_{n\to\infty} \frac{E[\#islands] - E[\#pools]}{n} = \lim_{n\to\infty} \frac{p^n - q^n}{n} = 0$$

4.3 The General Case

Our $m \times n$ torus can be regarded as m rows with each row consisting of a circular array of length n. For each of the m circular arrays, the expected number of islands is as given in the previous section. Then one would hope that, using the linearity of expectation, the expected number of islands is

$$m(npq + p^n)$$

by summing over all rows. But this is wrong! We must account for the fact that when an island in row $i + 1$ overlaps with another in row i (i.e. $a[i, j] =$

$a[i+1, j] = 1$ for some j), only one of them must be counted. This can be adjusted for by subtracting one for that overlap.

Therefore, we have to subtract from the above expression the expected number of overlaps. Let us consider this idea more carefully. As we move through the rows in order,

- If an island in row $i+1$ overlaps with **zero** islands in row i, then it is the start of a new island on the torus, and hence should be counted.
- If an island in row $i+1$ overlaps with **one** island in row i, then subtracting one for that overlap correctly adjusts for the number of islands, as both islands become part of the same island on the torus.
- If an island in row $i+1$ overlaps with **many** islands in row i, then we have two cases to consider: if those islands in row i are not part of the same island on the torus, then subtracting the number of overlaps, as before, correctly adjusts for the number of islands. If, however, some of those islands in row i are part of the same island on the torus (due to previous overlaps), then subtracting the number of overlaps adjusts for islands and subtracts the number of pools within the newly formed island.

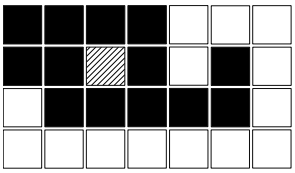

Figure 5: The cases described above. We count one island in the first row. When we reach the second row, we count another three, for a total of four islands. But each of the first two islands in the second row will subtract one overlap, bringing the number of islands down to two. By the time we reach the third row, we have one additional island, making the number of islands equal to three. That new island subtracts three overlaps (two of them with the same island), bringing the number down to zero, which is one island minus one pool (dashed) formed within the island.

Based on the above description, subtracting the expected number of overlaps gives $E[\#islands - \#pools]$, which by the linearity of expectation is equal to $E[\#islands] - E[\#pools]$. This, however, does not account for everything! An island that circles the torus through all the rows contributes an extra overlap (with itself) and, hence, ends up canceling itself. But we only have $O(n)$ such islands. Similarly, pools that are not contained within islands are not counted. But we cannot have more than $O(m+n)$ such unbounded pools. Consequently, our approach will be off by only $O(m+n)$ in counting $E[\#islands] - E[\#pools]$.

Therefore, if $\#overlaps$ denotes the number of overlaps, as m and n go to infinity:

$$\lim_{m,n\to\infty} \frac{E[\#islands] - E[\#pools]}{mn} = pq - \lim_{m,n\to\infty} \frac{E[\#overlaps]}{mn}$$

To obtain $E[\#overlaps]$, define the following indicator random variables: X_{ij} is an indicator random variable for the event that $a[i,j]$ is the edge of an island in row i, Y_i for the event $a[i,0] = a[i,1] = \ldots = a[i,n-1] = 1$, and Z_{ijk} for the event that an island in row i with edge $a[i,j]$ overlaps an island in row $i+1$ with edge $a[i+1,k]$. Observe that the X's, Y's, and Z's are pairwise independent in different rows.

$$\#overlaps = \sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\sum_{k=0}^{n-1} Z_{ijk} + \sum_{i=0}^{m-1} Y_i \sum_{j=0}^{n-1}(X_{(i-1)j} + X_{(i+1)j}) + \sum_{i=0}^{m-1} Y_i Y_{i+1}$$

Now, $P(X_{ij} = 1) = pq$ and $P(Y_i = 1) = p^n$. To make an overlap between two islands with edges $a[i,j]$ and $a[i+1,k]$, assume without loss of generality that $j \geq k$; then is it necessary that $a[i,j-1] = \ldots = a[i,k] = 1$ or $a[i+1,k-1] = \ldots = a[i+1,0] = a[i+1,n-1] = \ldots = a[i+1,j] = 1$. In general, one sequence will have length $(j-k) \bmod n$ and the other length $n-(j-k) \bmod n$. Therefore, $P(Z_{ijk} = 1)$ is obtained by multiplying $P(X_{ij} = 1)P(X_{(i+1)k} = 1)$ by $(p^d + p^{n-d} - p^n)$ when $1 < d < n-1$ (and by 1 when $d = 0$ and by p when $d = 1$ and $d = n-1$), where $d = (j-k) \bmod n$ (using $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$). By the linearity of expectation and the independence of the random variables,

$$E[\#overlaps] = \sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\sum_{k=0}^{n-1} E[Z_{ijk}] + 2\sum_{i=0}^{m-1}\sum_{j=0}^{n-1} E[Y_i]E[X_{(i+1)j}] + \sum_{i=0}^{m-1} E[Y_i]E[Y_{i+1}]$$

$$= \sum_{i=0}^{m-1}\sum_{j=0}^{n-1} p^2q^2\Big(1 + p + p + \sum_{d=2}^{n-2}(p^d + p^{n-d} - p^n)\Big) + 2mnqp^{n+1} + mp^{2n}$$

$$= mnp^2q^2\Big(1 + p + p + \sum_{d=2}^{n-2}(p^d + p^{n-d})\Big) - mn(n-3)q^2p^{n+2} + 2mnqp^{n+1} + mp^{2n}$$

Therefore, for $p < 1$ (but the result is also valid for $p = 1$),

$$\lim_{m,n\to\infty} \frac{E[\#overlaps]}{mn} = p^2q^2\Big(\sum_{d=0}^{\infty} p^d + \sum_{d=1}^{\infty} p^d\Big) = p^2q(1+p)$$

and by substituting,

$$\lim_{m,n\to\infty} \frac{E[\#islands] - E[\#pools]}{mn} = pq - p^2q(1+p) = pq(q - p^2)$$

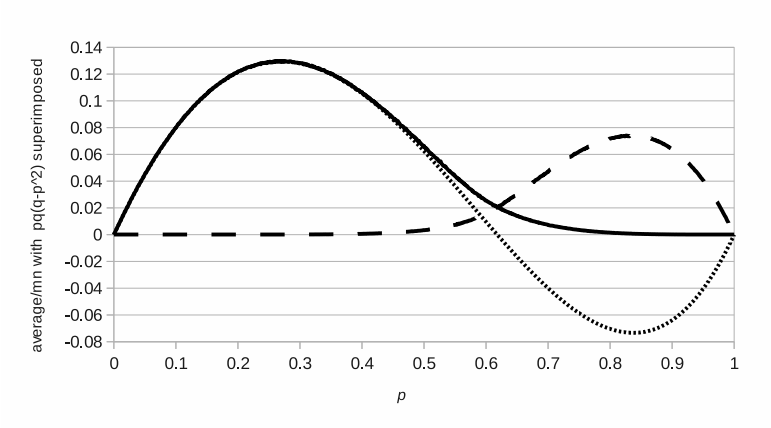

Figure 6: Average number of islands (solid) and pools (dashed) divided by torus size mn with $pq(q-p^2)$ (dotted) superimposed as a function of p.

4.4 The Golden Ratio

The limit derived above is trivially zero when $p = 0$ ($E[\#islands] = 0$ and $E[\#pools] = 1$) or $q = 0$ ($E[\#islands] = 1$ and $E[\#pools] = 0$). It is also zero when

$$q = p^2$$
$$\frac{1}{p} = \frac{p}{q}$$
$$\frac{p+q}{p} = \frac{p}{q}$$

which is true if and only if $p/q = \phi = (1+\sqrt{5})/2$, the well celebrated golden ratio (i.e. $p = 1/\phi \approx 0.618$). Observe also that $pq(q-p^2)$ changes sign from positive to negative at $p = 1/\phi$. In addition, islands will tend to merge when p is high, and pools will tend to merge when q is high. Therefore, for large enough m and n,

$$\frac{E[\#islands]}{mn} \approx pq(q-p^2), E[\#pools] \ll E[\#islands] \qquad \frac{p}{q} \ll \phi$$

$$E[\#islands] = E[\#pools] \qquad \frac{p}{q} = \phi$$

$$\frac{E[\#pools]}{mn} \approx pq(p^2-q), E[\#islands] \ll E[\#pools] \qquad \frac{p}{q} \gg \phi$$

This conclusion is consistent with Figure 6 and reveals an interesting property of balance between islands and pools (in the expected sense) when p/q is the golden ratio.

5 Conclusion

I started this exposition with a programming perspective to show how the introduction of the modulo operator can facilitate the handling of two-dimensional arrays. The two-dimensional array became a torus and an intriguing question arose about the number of islands and pools on a random torus. This lead to a probabilistic analysis that revealed an interesting link between the expected numbers of islands and pools and the golden ratio. In performing this analysis, the most theoretical sophistication I used was that expectation is linear, i.e. $E[X+Y] = E[X]+E[Y]$. I presume that college students, whether in computer science or mathematics, should be comfortable in following this exposition. So cheers for the linearity of expectation, and more cheers for mathematics, which provide us with the insight to uncover the realities of our world. Well, in this particular case, a fictitious world!

6 Notes

A general treatment of the subject of cluster formation in lattices can be found in percolation theory [1, 2]. To the best of my knowledge, this particular setting of islands and pools with a link to the golden ratio is unique. The golden ratio itself has been the subject of many studies (and fallacies, see [3] for references); and despite all the arguments about its inherent beauty, my use of it is simply restricted to the mathematical fact itself. The linearity of expectation and the use of indicator random variables are typical topics in probability textbooks, but [4] provides a concise description with examples. Finally, for a flavor of what it might be like to live on a torus, I refer the reader to Ringworld [5], a science fiction idea introduced in the seventies.

References

[1] Dietrich Stauffer and Ammon Aharony: Introduction to Percolation Theory. CEC Press (1994).

[2] Béla Bollobás and Oliver Riordan: Percolation. Cambridge University Press (2009).

[3] Clement Falbo: The Golden Ratio, A Contrary Viewpoint. The College Mathematics Journal 36(2) (2005) 123-134.

[4] Thomans Cormen, Charles Leiserson, Ronald Rivest, and Clifford Stein: Introduction to Algorithms. MIT Press (2001).

[5] Larry Niven: Ringworld. Random House Publishing Group (1970).

Appendix: Pseudocode for Counting Islands and Pools

Here's a pseudocode for counting islands and pools. For correctness, the programming language used must return a non-negative integer for the modulo operator. But many don't when $x < 0$ in $x \bmod n$, so a fix can replace $x \bmod n$ with $(x + n) \bmod n$ to avoid a negative x.

```
VISIT(a, i, j, b)
  if a[i, j] = b ∧ ¬visited[i, j]
    then visited[i, j] ← TRUE                          ▷ mark it as visited
         VISIT(a, i, (j − 1) mod n, b)                 ▷ recurse through neighbors
         VISIT(a, i, (j + 1) mod n, b)
         VISIT(a, (i − 1) mod m, j, b)
         VISIT(a, (i + 1) mod m, j, b)
         if b = 0
           then VISIT(a, (i − 1) mod m, (j − 1) mod n, 0)   ▷ and diagonally for pools
                VISIT(a, (i − 1) mod m, (j + 1) mod n, 0)
                VISIT(a, (i + 1) mod m, (j − 1) mod n, 0)
                VISIT(a, (i + 1) mod m, (j + 1) mod n, 0)


COUNT(a, m, n, b)
   for i ← 0 to m − 1
     for j ← 0 to n − 1
       visited[i, j] ← FALSE
   total ← 0
   for i ← 0 to m − 1
     for j ← 0 to n − 1
       if a[i, j] = b ∧ ¬visited[i, j]                 ▷ when seen for the first time
         then VISIT(a, i, j, b)                        ▷ visit the rest of it
              total ← total + 1                        ▷ and increment the count
   return total


ISLANDS(a, m, n)
  return COUNT(a, m, n, 1)

POOLS(a, m, n)
   return COUNT(a, m, n, 0)
```