Fibonacci in The Curriculum: Not Just a Bad Recurrence *

Saad Mneimneh
Computer Science
Hunter College and the Graduate Center of the
City University of New York (CUNY)
New York, USA
saad@hunter.cuny.edu

ABSTRACT

As an advocate of infusing various algorithmic and mathematical aspects when teaching about programming, I have come to realize that an early such practice is essential for a rounded computer science education. In this paper, I show how this can be done while focusing on one theme: Fibonacci.

Perhaps the most common use of Fibonacci has been to show the power of recurrence in implementing the Fibonacci sequence, which is often accompanied by a caveat that it is not the best implementation (very slow). Nevertheless, the sequence, with its rabbit story and celebrated golden ratio, is a rather exciting "gadget" for many students and it often pays off to introduce it. Therefore, I explore ways to use Fibonacci (the binary word) and the golden ratio for guiding implementation, and to successfully convey an important message of computer science that programming is not just about writing code. This will be done in the context of one dimensional and two dimensional arrays.

1. PART I: SKOLEM GOES DOWN THE RABBIT WHOLE

When teaching about one dimensional arrays, it is often natural to think about sequences. A Skolem sequence, named after its creator Thoralf Skolem in 1957 [5], is a sequence of $2k$ integers $s_1, \ldots, s_{2k}$ such that for every $n \in \{1, \ldots, k\}$ there exist two integers $a_n < b_n$ that satisfy $s_{a_n} = s_{b_n} = b_n - a_n = n$. Interestingly, a closely related formulation was made in parallel by C. Dudley Langford in 1958 after observing his child play with colored cubes [2]. Stories like this one make a good introduction and provide some context when talking about abstract problems. That was among several motivations (including the creation of a fresh problem) in considering Skolem sequences. A Skolem sequence for $k = 4$ is shown below:

1 1 3 4 2 3 2 4

Figure 1: A Skolem sequence for $k = 4$: $s_1 = s_2 = 2 - 1 = 1$, $s_5 = s_7 = 7 - 5 = 2$, $s_3 = s_6 = 6 - 3 = 3$, and $s_4 = s_8 = 8 - 4 = 4$.

1.1 The Birth of an Infinite Skolem Squenence

Not every finite k admits a Skolem sequence, and when it does, it is far from trivial to construct one. This lead me to define an infinite Skolem sequence.

DEFINITION 1.1. *(Infinite Skolem Sequence) An infinite Skolem sequence $\{s_i : i \in \mathbb{N}\}$ is such that for every $n \in \mathbb{N}$ there exists exactly one pair of integers $a_n < b_n$ that satisfy $s_{a_n} = s_{b_n} = n$. Furthermore $b_n - a_n = n$.*

As I will illustrate below, constructing an infinite Skolem sequence is much easier.

1.2 The First Lexicographic Infinite Skolem Sequence

It is trivial to construct an infinite Skolem sequence because we can always "fill in the gaps"; for instance, with the smallest unused number (see Figure 2). Furthermore, one could refer to this as the first infinite Skolem sequence in a lexicographic order. Students will appreciate this terminology when they know it is nothing but the dictionary order. In fact, an entertaining exercise would be to prove that the infinite Skolem sequence thus constructed is indeed the first in a lexicographic order. Mathematically, this is equivalent to say $n < m \Leftrightarrow a_n < a_m \Leftrightarrow b_n < b_m$.

In what follows, any reference to the infinite sequence signifies this first lexicographic (thus unique) infinite Skolem sequence. As it turns out, this particular Skolem sequence, given by (a_n, b_n) for every $n \in \mathbb{N}$, exists in the literature under the name Wythoff pairs [6], and has been the subject of study in many publications of the Fibonacci Quarterly. Let us now discover what it looks like!

1.3 Discovering Skolem: A Warm Up

An immediately obvious candidate for a warm up question is to design an algorithm that generates the first, say k,

*The theoretical results reported in this paper were obtained by the author during the process of designing homework questions for an introductory course in programming. Although this work lies on the boundary of education and research, the author has determined that this publication is not the proper venue for sharing the mathematical proofs. The proofs are elementary, but they are not strongly relevant for the educational merit of the exposition.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SIGCSE'15, March 4–7, 2015, Kansas City, MO, USA.
Copyright © 2015 ACM 978-1-4503-2966-8/15/03 ...$15.00.
http://dx.doi.org/10.1145/2676723.2677215.

RIGHTSLINK

1	1	_	_	_	_	_	_	_	_	_	_	_	_	_
1	1	2	_	2	_	_	_	_	_	_	_	_	_	_
1	1	2	3	2	_	3	_	_	_	_	_	_	_	_
1	1	2	3	2	4	3	_	_	4	_	_	_	_	_
1	1	2	3	2	4	3	5	_	4	_	_	5	_	_
1	1	2	3	2	4	3	5	6	4	_	_	5	_	6
1	1	2	3	2	4	3	5	6	4	7	_	5	_	6
1	1	2	3	2	4	3	5	6	4	7	8	5	_	6
1	1	2	3	2	4	3	5	6	4	7	8	5	9	6

Figure 2: The leading 15 terms of the first lexicographic infinite Skolem sequence.

integers of this infinite sequence. Why is this an important exercise? Unlike its Fibonacci counterpart, this exercise is not possible without using additional memory (well, at least for now), the kind that only arrays (or more advanced structures) can provide. A typical solution would create an array of the appropriate size k and navigate through it to assign all of its elements. An algorithm is shown in Figure 3 (in pseudocode).

```
for i ← 1...k
  do s[i] ← 0   ▷ gaps
i ← 1
n ← 0
while i ≤ k
    do n ← n + 1   ▷ next number
       s[i] ← n
       if i + n ≤ k   ▷ does it fit?
          then s[i + n] ← n
       while s[i] ≠ 0 ∧ i ≤ k   ▷ find next gap
          do i ← i + 1
```

Figure 3: Algorithm for constructing $s_1, \ldots, s_k$, n will be the largest integer in the sequence.

I list below some aspects that make the implementation tricky and, therefore, worthy of being a non-trivial exercise:

- there is a nested loop
- one needs to keep track of gaps
- the value to be assigned and the array index move at different paces
- one has to use caution when indexing the array, especially when assigning the second value of each pair (which may not fit)

1.4 A Real Programming Challenge

My real interest in the Skolem problem is the following: Given n, find a_n, i.e. the index of the first occurrence of n in the infinite sequence. For example, if $n = 9$, then the answer is 14 (see Figure 2). The reason for my particular interest is to eventually shed some light on an alternative solution for the construction of Figure 3. But more importantly now, to present the student with the typical situation where one must use an array without knowing its size (not even at run time, so this is not about dynamic memory allocation). In Figure 3, when s_i is assigned the value n, $a_n = i$ is automatically revealed. However, it may not be obvious to the student how to choose the size of the array to guarantee $k \geq a_n$ (otherwise, n will not show up). This situation can be avoided with the use of advanced data structures; however, I am excluding this kind of knowledge for beginners. Therefore, the first impression is that this situation cannot be avoided. Consequently, a wishful attempt would be to perform the algorithm of Figure 3 and hope that the desired n will show up (see Figure 4)!

```
Find(n)
  for i ← 1...k
    do s[i] ← 0   ▷ gaps
  i ← 1
  m ← 0
  while i ≤ k
      do m ← m + 1   ▷ next number
         if m = n
           then return i   ▷ this is a_n
         s[i] ← m
         if i + m ≤ k   ▷ does it fit?
           then s[i + m] ← m
         while s[i] ≠ 0 ∧ i ≤ k   ▷ find next gap
           do i ← i + 1
  ▷ now what?!
```

Figure 4: A wishful thinking approach.

The approach illustrated in Figure 4 is not a bad idea for a start. Based on this implementation, one could repeatedly perform the entire process with a larger k until n shows up. This will be correct, but a bit clumsy. A better approach is to guide the students to figure out an upper bound on k. For instance, when $m = n$ in Figure 4, we know that we must have written at most $2(n-1)$ values into the array (m is assigned at most twice to some $s[i]$). Therefore, $a_n \leq 2(n-1)+1 = 2n-1$, and it suffices to make $k = 2n-1$ to guarantee that m will reach n. But we can do better.

1.5 Follow the RabBIT

At this juncture, it is probably a good idea to stop writing code and think. I usually provide the students with some strategy to guide their thoughts and explore the problem. This is what lead me to establish the following interesting theorem about Skolem and the infinite (binary) Fibonacci word [4] (stated without proof, other forms of this theorem are a ready consequence of some classical results about Wythoff pairs and the positions of the n^{th} 1 and the n^{th} 0 in the infinite Fibonacci word): [1]

THEOREM 1.2. *$a_n = n + \sum_{i=1}^{n-1} f(i)$ where $f(n)$ is the n^{th} bit of the infinite Fibonacci word.*

The students are encouraged to explore the problem by studying the behavior of $a_n - a_{n-1}$. This is relevant be-

[1]The infinite Fibonacci word can be constructed by writing 10, then scanning the bits in order starting with the first, and appending the sequence with 110 upon seeing a 1, and 10 upon seeing a 0. The first few bits are as follows: 1011010110 ...

RIGHTSLINK

cause of the following equality:

$$a_n = a_1 + (a_2 - a_1) + (a_3 - a_2) + \ldots + (a_n - a_{n-1})$$

The algorithm of Figure 3 can be easily modified to compute this differential, and upon doing so, one can observe that $a_n - a_{n-1}$ seems to always be either 1 or 2.

$$\begin{aligned} a_2 - a_1 &= 2 \\ a_3 - a_2 &= 1 \\ a_4 - a_3 &= 2 \\ a_5 - a_4 &= 2 \\ a_6 - a_5 &= 1 \\ a_7 - a_6 &= 2 \\ a_8 - a_7 &= 1 \\ a_9 - a_8 &= 2 \\ &\vdots \end{aligned}$$

The good student will hopefully realize that $a_n - a_{n-1}$ is in effect binary (two valued function of n), and that this fact can be expressed in the following way:

$$\begin{aligned} a_2 - a_1 - 1 &= 1 \\ a_3 - a_2 - 1 &= 0 \\ a_4 - a_3 - 1 &= 1 \\ a_5 - a_4 - 1 &= 1 \\ a_6 - a_5 - 1 &= 0 \\ a_7 - a_6 - 1 &= 1 \\ a_8 - a_7 - 1 &= 0 \\ a_9 - a_8 - 1 &= 1 \\ &\vdots \\ a_n - a_{n-1} - 1 &= f(n-1) \end{aligned}$$

Upon making this observation, and with a hint to the infinite Fibonacci word, a student should be able to formulate Theorem 1.2 (without proving it, simply by adding up all the rows and using the fact that $a_1 = 1$). This in turn should provide the idea for a better solution for our challenge because we only require the first $n-1$ bits of infinite Fibonacci word, hence we can work with an array of size $n-1$ (Figure 5).

```
Find(n)
  compute f[1], ..., f[n − 1]   ▷ a nice exercise by itself
  sum ← 0
  for i ← 1 to n − 1
    do sum ← sum + f[i]
  return n + sum
```

Figure 5: Algorithm to find a_n given n.

In fact, the need for an array can be eliminated entirely. A deeper understanding of the infinite Fibonacci word will reveal that:

$$\sum_{i=1}^{n} f_i = \left\lfloor \frac{n+1}{\phi} \right\rfloor$$

where $\lfloor x \rfloor$ is the largest integer $\leq x$, and $\phi = (1+\sqrt{5})/2$ is the golden ratio! Therefore, $a_n = n + \lfloor n/\phi \rfloor = \lfloor n(1+1/\phi) \rfloor = \lfloor n\phi \rfloor$ (and $b_n = n + \lfloor n\phi \rfloor = \lfloor n(1+\phi) \rfloor = \lfloor n\phi^2 \rfloor$), which was proved by Wythoff in [6]. It now becomes interesting to redo the algorithm of Figure 3 without using an array, but I will leave this potential open for imagination.

From the conceptual perspective, this marks the end of the journey to the rabbit hole, but one could offer guidance throughout this journey with various levels of detail. The important points are:

- we start with a warm up question
- we follow it by a challenge
- the challenge could be solved by adapting the warm up solution
- the students are guided to study the problem in a specific way
- they are also either assumed to know about the infinite Fibonacci word or to be told to learn about it
- upon making the connection, the students should be able to come up with a better solution for the challenge

2. PART II: LIVING ON A RANDOM TORUS

We don't see doughnut shapes when we look up in the sky. So it takes a spark of imagination to consider the possibility of living on a planet in the shape of a torus. 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 $n \in \mathbb{N}$).

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

The modulo operator is widely recognized as the remainder (an integer) in the division of x by n, ranging from 0 to $n-1$. Surprisingly, in 1970 Larry Niven has explored the possibility of living on a toroidal world in Ringworld [3]. Again, this makes an interesting opening to introduce the students to the problem discussed herein.

2.1 The Birth of a Torus

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, even the beginner 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 6. The definitions in Figure 6 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 7.

The torus is born, and all that remains to be done is adding some life to it! Hence, for an initial 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.

RIGHTSLINK

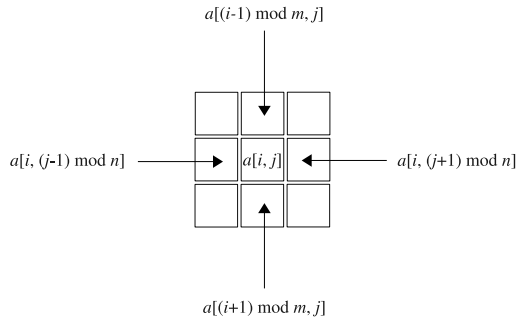

Figure 6: Every $a[i, j]$ has all the neighbors.

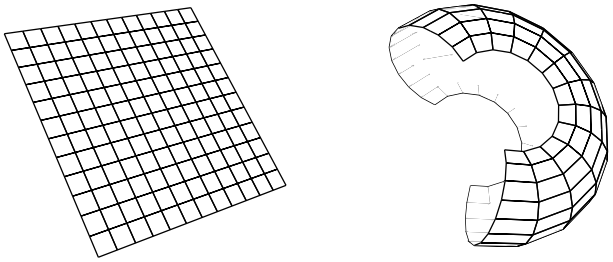

Figure 7: Folding a two-dimensional array into a torus. On the left, the array, and on the right, the folding in action showing a partially folded torus.

2.2 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 6.

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 2.1 (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. 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 exercise 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 (see following section).

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 2.2 (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 8 shows an example of this formation on a random torus.

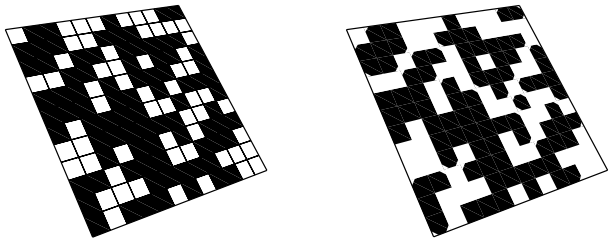

Figure 8: From a two-dimensional array to islands and pools on the torus, showing a total of five islands (black) and two pools (white). Avoid the overcounting of islands and pools when neighbors wrap around in modulo m and/or n.

2.3 Let's Count: A Warm Up

The warm up question here is of course to count islands and pools. This problem is closely related to the flood filling performed by the "bucket" tool of graphics editors, which relies on finding connected components, a standard algorithmic topic. A pseudocode for counting islands and pools on a torus is shown in Figure 9. The code is self explanatory and assumes that students are familiar with recursion. But since recursion is often tricky, this makes it worthy of a nontrivial exercise: 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.

2.4 A Real Programming Challenge

A question that intrigued me is the following: Consider a random torus where p is the probability of land (and hence $q = 1 - p$ is the probability of water). For what value of p is

RIGHTSLINK

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


Count(b)   ▷ b is 1 for islands and 0 for pools
  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]   ▷ first time seen
        then Visit(i, j, b)   ▷ visit the rest of it
             total ← total + 1   ▷ and increase the count
  return total
```

Figure 9: Pseudocode for counting islands and pools. For correctness, the programming language 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.

the number of islands equal to the number of pools (in the average sense of course)? As in Part I, the solution for the warm up question can be adapted to answer the challenge. A straightforward approach would be to compute for every p the average numbers of islands and pools, and observe when they are approximately equal. A thought experiment is shown in Figure 10 in pseudocode.

```
Try(p)
  avgI ← 0   ▷ average number of islands
  avgP ← 0   ▷ average number of pools
  for i ← 1 to T
    initialize the torus randomly using p
    avgI ← avgI+Count(1)/T
    avgP ← avgP+Count(0)/T
  return avgI − avgP
```

Figure 10: A thought experiment.

My interest in this question, beside it being experimental leading to a result, follows from a simple fact: often a seemingly correct concept can fail to yield the desired result. The thought experiment of Figure 10 is conceptually correct, i.e. repeated use of it with different values of p should reveal the value of p that makes $avgI - avgP \approx 0$. Wrong! And here's why:

- m and n must be large enough to produce the desired effect
- It is not obvious how to try different values for p and what granularity will guarantee to capture the desired value.
- Almost no value of p will make $avgI - avgP$ approximately 0, as a slight discrepancy in the number of islands and pools is magnified by the torus size mn.

Nevertheless, one could still rely on the algorithm of Figure 10 to discover, for instance, when $avgI - avgP$ switches sign. Still, it's a clumsy method and it will only give an approximate range for the desired value of p, which in turn may fail to be identified as any special value. To rectify this problem, the students are encouraged to explore $avgI - avgP$ scaled by the torus size as a function of $p/q = p/(1-p)$ (an alternative is to study $avgI/avgP$ to eliminate the effect of the torus size, but this quantity is unbounded, and it is better to work with a bounded quantity, see below).

2.5 Ringworld*ians* discover the golden ratio!

Humans have always been fascinated by the golden ratio, to the point of abuse [1]. Similarly, the inhabitants of Ringworld will have discovered that divine ratio simply by observing their own world. As it turns out, an attempt to avoid the problems outlined in the previous section leads to the following result (again, stated without proof):

THEOREM 2.3. *Let $avgI$ be the average number of islands and $avgP$ be the average number of pools, then:*

$$\lim_{m,n\to\infty} \frac{avgI - avgP}{mn} = pq(q - p^2)$$

The theorem has several consequences. For large m and n, $avgI - avgP \approx mnpq(q - p^2)$. This also means that $avgI \approx mnpq(q-p^2)$ when p is small, as the number of pools will tend to be small compared to the number of islands. A symmetric argument shows that $avgP \approx mnpq(p^2-q)$ when p is large. Most importantly, and to quantify what a small or large p would be, the theorem shows that $avgI = avgP$ when $q = p^2$, and since $q = 1 - p$, this means $p/q = \phi$, the golden ratio! Therefore, a small p satisfies $p/q \ll \phi$, and a large p satisfies $p/q \gg \phi$.

COROLLARY 2.4 (GOLDEN TORUS). *For large m and n,*

$$avgI \approx mnpq(q - p^2) \quad p/q \ll \phi$$

$$avgI = avgP \qquad p/q = \phi$$

$$avgP \approx mnpq(p^2 - q) \quad p/q \gg \phi$$

where $\phi = (1 + \sqrt{5})/2$ is the golden ratio.

A proper guidance to explore p/q, with a hint to the golden ratio, can improve the attempt of Figure 10 as shown in Figure 11.

It may still be inconvenient to try several values of p, but the range can now be controlled in a pragmatic way through the use of ϵ. Moreover, students can be guided to try

RIGHTSLINK

```
Try(p, ε)
  avgI ← 0   ▷ average number of islands
  avgP ← 0   ▷ average number of pools
  for i ← 1 to T
    initialize a large m × n torus randomly using p
    avgI ← avgI+Count(1)/T
    avgP ← avgP+Count(0)/T
  if −ε < (avgI−avgP)/mn < ε
    then return p/(1 − p)
    else return −1
```

Figure 11: An improved experiment.

values of p that are a ratio of consecutive Fibonacci numbers. When m and n are large enough (say $m = n = 100$) and ϵ is small enough, the above program will return a number close to the golden ratio (which is recognizable) or -1 (a failure for the given p). A student can then set $p/q = \phi$, accordingly $p = 1/\phi$, and observe that $\frac{avgI}{mn}$ and $\frac{avgP}{mn}$ are approximately 0.0205. But the 0.0205 is still a mystery to me.

And since silence is also golden, I will stop here by repeating a similar summary for this outer space journey:

- we start with a warm up question
- we follow it by a challenge
- the challenge could be solved by adapting the warm up solution
- the students are guided to study the problem in a specific way
- they are also either assumed to know about the golden ratio or to be told to learn about it
- upon making the connection, the students should be able to come up with a better solution for the challenge

3. A NOTE ON THE TWO PROBLEMS

While it is not hard to believe that $p/q = \phi$ achieves the desired balance of islands and pools on the torus (it can be verified), it takes a leap of faith to believe that $a_n = n + \sum_{i=1}^{n-1} f(i)$ for all $n \in \mathbb{N}$ without a mathematical proof. Given the approach outline herein, however, students might not actually prove any mathematical result, though such a direction can definitely be pursued with the better students. The main goal of this approach remains to be the exposure to mathematical/algorithmic context.

4. CONCLUSION

In this exposition, I showed how to infuse some algorithmic and mathematical aspects to guide the programming experience. The main theme is Fibonacci (and the golden ratio), which is a pleasant topic for many students. The typical paradigm that I support here is to first start with a warm up question (one that is not too trivial), then to follow up on it with a programming challenge that can be solved by adapting the warm up solution, but not to a satisfactory level. The students are then encouraged through hints to explore the problem in a certain way, including the introduction of some mathematical concepts. Finally, upon making the connection between the hints and the original problem, the students are expected to figure out a better solution for the challenge.

5. THE AFTERMATH

Over the years, many of my students end up appreciating the approach outlined above. Students enter the field with the impression that computer science is about the ability to write programs, and that programming is a simple mechanical task. They later hear differently from their peers, those who have had me as their instructor. And while they understand that I will be pulling them out of their comfort zone, students seek to be placed in my sections, as I have managed to create a particular reputation for my teaching. This remains true even for average students who do not actively look for an extra challenge. They opt to pursue this path because they feel that they will be learning something interesting.

It takes a while to understand the philosophy of the kind of challenging, but at the same time guided, assignments. Initially, the students may feel lost in terms of what is being required from them, but they eventually get the hang of it once they become comfortable with my style. They build a level of independence over time, and I have observed that, towards the end of the semester, only a few students will gather at my door.

On a different note, I should say that my approach pays off even when a student is not academically savvy but career oriented. For one thing, many respectful software companies, e.g. Google, follow a similar approach for their interview process. They expect the applicant to first solve a given problem in the most obvious way, with no regards to the efficiency or the elegance of the solution. They then raise the level by guiding the applicant to pursue a certain direction of thought, hopefully leading to a better solution.

Finally, I think we are in great need, more than any time, to rescue the declining mathematical level in computer science education. In my exposition, I illustrated specific examples of how to expose students to contexts that are mathematical and algorithmic in nature. But I hope that I was able to convey the general approach of exploring such a strategy as a guide for writing programs and obtaining solutions.

6. REFERENCES

[1] C. Falbo. The golden ratio: A contrary viewpoint. *The College Mathematics Journal*, 36(2):123–134, 2005.

[2] C. D. Langford. Problem. *Mathematical Gazette*, 42:228, 1958.

[3] L. Niven. Ringworld. *Random House Publishing Group*, 1970.

[4] OEIS. The online encyclopedia of integer sequences. *http://oeis.org/A005614*.

[5] T. Skolem. On certain distributions of integers in pairs with given differences. *Mathematica Scandinavica*, 5:57–58, 1957.

[6] W. A. Wythoff. A modification of the game of nim. *Nieuw Archief voor WisKunde*, 7(2):199–202, 1907.

RIGHTSLINK