Math CountS
Penny Piles

Peter B. Henderson
Department of Computer Science & Software Engineering
Butler University
Indianapolis, Indiana 46208 USA
<phenders@butler.edu>

Problem 1: Given 10 pennies, how many different ways can you create piles on a table, or stacks, with them? Two obvious answers at each end of the spectrum are one pile of 10 pennies and 10 piles of one penny. When given a problem, students should learn to look for the obvious and simple first, even if this entails simplifying the problem to get started or better understand it. Indeed, getting started is frequently an impediment for many students. Engage yourself, do something, anything!

Some strategies students might use for this problem are:

1) Brute force solution using 10 real pennies or 10 abstract pennies (e.g., circles or the letter ‘P’ on paper).
2) Observe that the number of piles can range from 1 to 10, with 1 and 10 “solved” above. Decompose the problem by considering 2, 3, 4, ..., 8, and 9 piles independently. Look for patterns or symmetry (e.g., perhaps the answer for 2 and 9 are the same).
3) Simplify by solving the problem for 1, then 2 pennies (real or abstract). Continue this process using inductive reasoning to seek a pattern. If you know there are k piles for n-1 pennies, how many are there for n pennies?
4) Reduce it to a known solved problem, perhaps using combinatorial analysis. Knowing some discrete math would help here.

Another approach might be to look for alternative representations of the problem to work with. For example, 10 might represent one pile of 10 pennies and 1,1,1,1,1,1,1,1,1,1 represents 10 piles of one penny.

Wait a minute, does the student understand the problem? What does “how many different ways” mean? Are the following two three piles of four pennies different¹?

```
P   P   P   P
P   P   P   P
```

In mathematical concepts is this problem dealing with bags \{1,1,2\} = \{1,2,1\} or tuples (1,1,2) \neq (1,2,1)? The problem must be clearly defined before continuing. For fun, try it both ways.

¹ How many times did you read this question?
In computer science one usually solves general problems. Given \( N > 0 \) pennies, how many different ways can you create piles, or stacks, with them? Can you think of a real life problem which matches this one? If so, please let me know.

That was a warm up to get you thinking about piles of pennies.

**Problem 2:** Given a sequence of piles of pennies on a table, remove one penny from each pile. Now, add a new pile to the right made from the pennies you removed. For example, the 4 piles of pennies on the left below become the 3 piles on the right.

```
P
P
P
P
P
P
P
```

Using tuples, an alternate representation for this transformation from one state to the next is:
\((1, 1, 2, 5) \rightarrow (1, 4, 4)\)

What is an obvious invariant of this state transformation? Hint: Conservation of pennies.

Recall an earlier Math Counts column dealing with using inductive reasoning and problem simplification to understand the problem, to identify general solution patterns, and to look for cases? Let’s “play” with this problem and see what we can learn.

1. Given the state \((1, 3, 2, 8, 6)\) what is the next state?

2. Is the case where no piles of pennies disallowed?

3. What constitutes a valid state? Answer: \((n_1, n_2, n_3, \ldots, n_k)\), \(k \geq 0\), \(\forall\ i = 1, 2, \ldots, k, n_i > 0\)

4. Given the state \((n_1, n_2, n_3, \ldots, n_k)\), \(k \geq 0\), \(\forall\ i = 1, 2, \ldots, k, n_i > 0\), how many pennies will there be in the rightmost pile of the next state?

5. Is the next state uniquely determined from any given state? (i.e., Is this transformation deterministic?)

6. What happens for the case when a pile has only one penny?

7. Given the state \((n_1, n_2, n_3, \ldots, n_k)\), \(k \geq 0\), \(\forall\ i = 1, 2, \ldots, k, n_i > 0\)

   a. Informally specify the next state (e.g., using natural language)
   b. Formally specify the next state using mathematics.

8. Starting with state \((1, 3)\) give the next 6 states.
9. For all starting states \(( n_1, n_2, n_3, \ldots, n_k )\), \( k \geq 0, \forall i = 1, 2, \ldots, k, n_i > 0\), after a finite number of transformations, will this starting state always occur again?

10. Give an argument supporting your answer to question 8.

11. Find a state which is invariant under this transformation. Hint: Simple states!

12. Present a general characterization of all states which are invariant under this transformation, and give an argument supporting your characterization.

Now that you understand the problem and have a basic solution strategy, you could program this transformation. Before doing so let’s make sure we really do have a good grasp by creating a simple model using a function definition language. This may seem like extra work (don’t we already understand and see the solution?), however, I have always found this endeavor useful, learning something new and relevant about the problem and solutions strategies.

Below is a function definition `transform` in Standard ML and a sample execution. Piles of pennies are represented by lists (e.g., \([ 1, 3 ]\)), ‘@’ is list concatenation, and \([ k ]\) creates a list with one element, \(k\). Note how the “structure” of this function definition matches the “structure” of the problem,

\[
\text{fun transform( piles ) = removeApenny( piles ) @ [ numberOf( piles ) ];}
\]

\[
> \text{transform( [5, 3, 2, 8, 1, 3, 1 ] );}
> \ [4, 2, 1, 7, 2, 7] : \text{int list}
\]

The definition of `removeApenny` uses pattern matching for cases and recursion. Operator ‘::’ prepends an element to a list (e.g., \(3 :: [ 4, 5 ] = [3, 4, 5]\)) and as an argument matches the pattern `first::rest` (e.g., for \([3, 4, 5]\) `first = 3` and `rest = [4, 5]`).

\[
\text{(* returns the sequence of piles of pennies resulting after removing *)}
\]

\[
\text{(* one penny from each pile in a given sequence of piles of pennies *)}
\]

\[
\text{fun removeApenny( [ ] ) = [ ]}
\]

\[
| \text{removeApenny( 1 :: rest ) = removeApenny( rest )}
\]

\[
| \text{removeApenny( n :: rest ) = (n - 1) :: removeApenny( rest );}
\]

\[
\text{(* returns the number of piles of pennies in a sequence of penny piles *)}
\]

\[
\text{(* function length returns the number of elements in a list *)}
\]

\[
\text{fun numberOf( piles ) = length( piles );}
\]
Below is a straightforward definition of a function which applies transform \( m > 0 \) times.

\[
\text{fun } m\_\text{transforms}(\text{piles}, 1) = \text{transform}(\text{piles}) \\
| m\_\text{transforms}(\text{piles}, m) = m\_\text{transforms}(\text{transform}(\text{piles}), m - 1); 
\]

Note: Students generally struggle with recursive reasoning. For more information check out David Ginat’s recent paper “Do Senior Students Capitalize on Recursion?,” in the Proceeding of ITiCSE 2004, Leeds, UK.

From what you have learned define the pre and post conditions for this problem, identify potential iteration invariants from the recursive definitions, and compose a program in your favorite imperative language.

Is there a more elegant, or efficient way, to compute \( m\_\text{transforms} \)? Some thoughts are:

- If \( m < n_1 \) then after \( m \) transformations the resulting pile will have \( m - n_1 \) pennies.
- Make use of the answer to (9) above to identify repetitions. That is, given a state \( (n_1, n_2, n_3, \ldots, n_k) \) what is the minimum number of transformations required to get back to this state?

Good luck and have fun!! Below is another problem with the same flavor. Indeed, if you have been reading David Ginat’s Inroads Colorful Challenges column you might see a connection between his matrix transmission problem (December 2003, page 25) and problem 3 below.

**Problem 3:** Consider a sequence of piles of pennies on a table. Each pile has zero or more pennies. Consider taking a penny from the leftmost non-empty pile and putting it in its adjacent rightmost pile. This is called a **move**. For example, starting with 2,3,2,0,8,6 one move would yield 1,4,2,0,8,6 and another move 0,5,2,0,8,6.

Create a similar sequence of discovery questions for this problem, play and see what you learn. Define a function **move** and then **m_moves**. For the latter, look for a more elegant/efficient strategy.

**Summary of recent events and activities related to mathematics in CS and SE education:**

1. **Nifty Applications in Discrete Mathematics** a Mathematical Association of America Professional Enhancement Program workshop organized by Bill Marion was held June 7-11, 2004 at Valparaiso University for mathematics and computer science faculty. Participants developed nifty problems in groups. When these are completed and evaluated, they will be posted at the URL [http://blue.butler.edu/~phenders/sigcse2004/niftyworkshop](http://blue.butler.edu/~phenders/sigcse2004/niftyworkshop) This is the URL for the Nifty Examples in Discrete Mathematics workshop Bill and I conducted at SIGCSE 2004.

---

2 Eventually you will see two possibilities – number of piles stays fixed or number of coins stays fixed. Your choice!!
If you have nifty discrete math problems you would like included, please send them to me.


Math CountS columns are available at http://blue.butler.edu/~phenders/InRoads/