The 7-11 Problem

Joshua E. Hill

e-mail: josh-math@untruth.org

Initial Release: November 6, 2010
Current Revision: August 26, 2011

1 THE PROBLEM

You walk into a 7-11 in Oregon (or any other mystical land where there is no sales tax) and grab four items from the shelf. Right as you are about to check out, the power goes off. The cashier indicates that it's no problem, as he has a calculator somewhere.

The cashier dusts off the calculator, and sets about totaling the four items. He finishes with a flourish, and then pauses and says, "Now that's odd… the total is \$7.11. Huh!"

You thoughtfully provide exact change, and as you are about to leave, the cashier gets a panicked look on his face and stops you.

"I'm really sorry, but I didn't total the products correctly! I accidentally multiplied all the prices together instead of adding them!"

You sigh, and put the items back on the counter, and the cashier again totals them up (making sure to use the "+" key this time). The cashier finishes and then looks distressed again. He says "That's odd! The total is \$7.11 again!"

Assuming that no additional errors were made in these two calculations and that no rounding occurred, what are the prices of the four items?

2 SOLUTIONS

The first thing that is important to note is that there are some restrictions: all the items must have a positive price, and that price must be an integer number of pennies.

If you don't make these assumptions, you don't get a finite number of answers. Indeed, these two relations in four unknowns yield a (2-dimensional) surface of valid answers in 3-dimensional space that looks something like Figure 1. In this graph, each axis is one of the item prices, and the fourth item's price is a function of the other three prices.

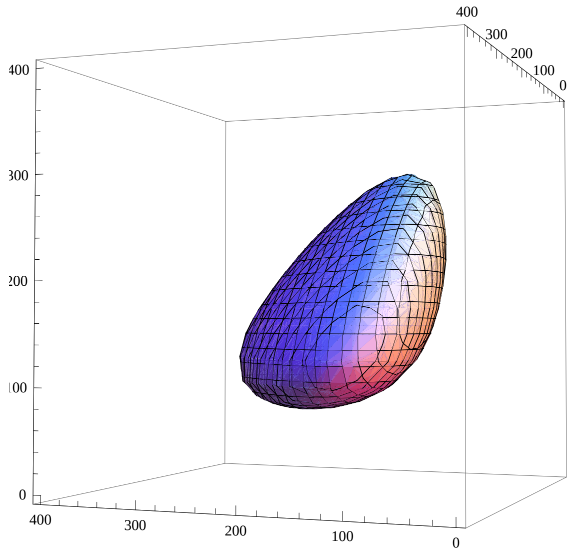

Figure 1: Graph of (mainly non-integer) results

To obtain the answer with no rounding to the nearest penny, we restrict ourselves to solutions that can be expressed as a positive integer number of pennies: If the prices are a, b, c, and d, our two relations are

$$\begin{aligned} a + b + c + d &= 7.11 \\ abcd &= 7.11 \end{aligned}$$

Let's instead look at the prices of the items in pennies: let $A = 100a$, $B = 100b$, $C = 100c$, $D = 100d$.

$$\begin{aligned} A + B + C + D &= 711 \\ ABCD &= 711000000 \end{aligned}$$

2.1 COMPUTATIONAL APPROACHES

Prior to putting serious thought into this problem, we could simply write a small program to solve it. We have two basic approaches:

- Additively by examining all four-element additive partitions of 711 and then look for values that multiply to $711,000,000$.
- Multiplicatively by examining all four-element factorizations of $711,000,000$ and then look for values that add to 711.

2.1.1 OUR FIRST ADDITIVE SOLUTION

We first iterate through every possible additive partition of 711 and check for ones that multiply to $711,000,000$. To remove solutions that are simply reordering of other solutions, we demand that $A \geq B \geq C \geq D$. This additive approach is described in Algorithm 1 (or Listing 1 in Section 3). It is very simple, but it is also the least efficient approach.

Algorithm 1: Additive Brute Force Approach

```
output: Integer Values $A$, $B$, $C$, $D$ such that
        $ABCD = 711000000$ and $A + B + C + D = 711$
$A \leftarrow 1$
while $A \leq 708$ do
    $B \leftarrow 1$
    while $A + B \leq 709$ and $B \leq A$ do
        $C \leftarrow 1$
        while $A + B + C \leq 710$ and $C \leq B$ do
            $D \leftarrow 711 - (A + B + C)$
            if $D \leq C$ then
                if $ABCD = 711000000$ then
                    return $A$, $B$, $C$, and $D$
                end if
            end if
            $C \leftarrow C + 1$
        end while
        $B \leftarrow B + 1$
    end while
    $A \leftarrow A + 1$
end while
```

This program produces a correct answer in $808,883$ inner loops and explores all the possibilities in $2,506,497$ inner loops. If you haven't become jaded with the supercomputer that likely resides on your desk, this sounds like an obnoxiously large calculation (but it resolves in less than a tenth of a second on a modern day machine).

Are we done? This approach is fundamentally inefficient, because it eventually explores each of the possible integer partitions of 711 into exactly four item prices, and most of these aren't really valid costs for the items (as most of these elements do not divide $711,000,000$). There are two approaches once we realize this: we can change our prior program to just check to see if each value is a divisor of $711,000,000$, or we can change the way that we enumerate so that we only get such values.

2.1.2 OUR SECOND ADDITIVE SOLUTION

Altering our first attempt so that each loop checks for divisibility is easy enough, as seen in Algorithm 2 (or Listing 2 in Section 3).

Algorithm 2: Additive Brute Force Approach (Try 2)

```
output: Integer Values $A$, $B$, $C$, $D$ such that
        $ABCD = 711000000$ and $A + B + C + D = 711$
$A \leftarrow 1$
while $A \leq 708$ do
    if 711000000 is divisible by $A$ then
        $B \leftarrow 1$
        while $A + B \leq 709$ and $B \leq A$ do
            if 711000000 is divisible by $B$ then
                $C \leftarrow 1$
                while $A + B + C \leq 710$ and $C \leq B$ do
                    $D \leftarrow 711 - (A + B + C)$
                    if $D \leq C$ then
                        if $ABCD = 711000000$ then
                            return $A, B, C, D$
                        end if
                    end if
                    $C \leftarrow C + 1$
                end while
            end if
            $B \leftarrow B + 1$
        end while
    end if
    $A \leftarrow A + 1$
end while
```

This approach runs in only 3,413 inner loops prior to finding the answer, and would run only 10,477 inner loops total if there were no match. That sounds great, but it tests for divisibility (a "mod" operation) quite a large number of times; 4,610 tests are required to get the answer. On most architectures, the mod operation is rather slow (e.g., approximately 32 times slower than multiplications on modern Intel processors). This relative expensiveness of the mod operation is also the reason that we don't check to make sure the C and D are divisors of 711,000,000. We'd have the false economy of trading two mod operations for five add/subtract operations, three multiplies and two conditional jumps, which is generally a losing proposition. We do check to make sure that A and B are divisors of 711,000,000, because that check prunes entire loops at great savings.

2.1.3 OUR FIRST MULTIPLICATIVE SOLUTION

Every price must be a divisor of $711,000,000$, that is must be of the form $2^i 3^j 5^k 79^\ell$, where $0 \leq i \leq 6, 0 \leq j \leq 2, 0 \leq k \leq 6, 0 \leq \ell \leq 1$. We don't want to find arbitrary divisors of $711,000,000$, instead we want to find only sets of divisors that multiply to $711,000,000$.

One way to do this is to recursively try every possible selection of such divisors, until it works out. Algorithm multRecA (or Listing 3 in Section 3) is such an approach.

Function multRecA

```
input : $n$ (the number of terms left), $d$ (the desired total),
        $\{i_{\max}, j_{\max}, k_{\max}, \ell_{\max}\}$
output: Returns "found" or "not found". If "found", the
        found value is output

if $n = 1$ then
    t $\leftarrow 2^{i_{\max}} 3^{j_{\max}} 5^{k_{\max}} 79^{\ell_{\max}}$
    if $t = d$ then
        Output $t$
        return *Found*
    end if
else
    $i_{\text{try}} \leftarrow 0$
    while $i_{try} \leq i_{max}$ do
        $j_{\text{try}} \leftarrow 0$
        while $j_{try} \leq j_{max}$ do
            $k_{\text{try}} \leftarrow 0$
            while $k_{try} \leq k_{max}$ do
                $\ell_{\text{try}} \leftarrow 0$
                while $\ell_{try} \leq \ell_{max}$ do
                    $t \leftarrow 2^{i_{\text{try}}} 3^{j_{\text{try}}} 5^{k_{\text{try}}} 79^{\ell_{\text{try}}}$
                    if $t \leq d - n + 1$ then
                        if multRecA($n-1$, $d-t$,
                        $\{i_{max} - i_{try}, j_{max} - j_{try}, k_{max} - k_{try}, \ell_{max} - \ell_{try}\}$)
                        =*found* then
                            Output $t$
                            return *found*
                        end if
                    end if
                    $\ell_{\text{try}} \leftarrow \ell_{\text{try}} + 1$
                end while
                $k_{\text{try}} \leftarrow k_{\text{try}} + 1$
            end while
            $j_{\text{try}} \leftarrow j_{\text{try}} + 1$
        end while
        $i_{\text{try}} \leftarrow i_{\text{try}} + 1$
    end while
end if
return *not found*
```

This approach is relatively simple, but not very efficient. It evaluates 9,844 different choices for A, B, C, and D prior to completing (it would evaluate $49,482$ different choices if there were no solution).

2.1.4 OUR SECOND MULTIPLICATIVE SOLUTION

The next approach is more complicated, but useful in better understanding the problem.

We first specify the recursive function at the center of this approach; this is basically the same as in the prior approach, except that it excludes the possible factor of 79. The only clever restriction that we make is marked "note (1)" in the multRecB function (or Listing 4 in Section 3), which allows us to exclude selections that do not fulfill the arithmetic/geometric inequality[1], which tells us that:

$$\left(\frac{\sum_{i=1}^{n} x_i}{n}\right)^n \geq \prod_{i=1}^{n} x_i$$

[1]`http://en.wikipedia.org/wiki/Inequality_of_arithmetic_and_geometric_means`

```
Function multRecB
    input  : $n$ (the number of terms left), $d$ (the desired total),
             $\{i_{\max}, j_{\max}, k_{\max}\}$
    output: Returns “found” or “not found”. If “found”, the
             found value is output
    if $n = 1$ then
        t $\leftarrow 2^{i_{\max}} 3^{j_{\max}} 5^{k_{\max}}$
        if $t = d$ then
            Output $t$
            return *Found*
        end if
    else
        if $d^n \geq 2^{i_{max}} 3^{j_{max}} 5^{k_{max}} n^n$ then            /* note (1) */
            $i_{\text{try}} \leftarrow 0$
            while $i_{try} \leq i_{max}$ do
                $j_{\text{try}} \leftarrow 0$
                while $j_{try} \leq j_{max}$ do
                    $k_{\text{try}} \leftarrow 0$
                    while $k_{try} \leq k_{max}$ do
                        $t \leftarrow 2^{i_{\text{try}}} 3^{j_{\text{try}}} 5^{k_{\text{try}}}$
                        if $t \leq d - n + 1$ then
                            if multRecB($n - 1$, $d - t$,
                            $\{i_{max} - i_{try}, j_{max} - j_{try}, k_{max} - k_{try}\}$) =
                            *found* then
                                Output $t$
                                return *found*
                            end if
                        end if
                        $k_{\text{try}} \leftarrow k_{\text{try}} + 1$
                    end while
                    $j_{\text{try}} \leftarrow j_{\text{try}} + 1$
                end while
                $i_{\text{try}} \leftarrow i_{\text{try}} + 1$
            end while
        end if
    end if
    return *not found*
```

Finally, we excluded the factor of 79 from the recursive step because we assume that the first item price has this factor. The main portion of the algorithm is in Algorithm 3 (or Listing 4 in Section 3) thus looks very much like the recursive step, but with an extra factor of 79.

Algorithm 3: Multiplicative Brute Force Approach II (main)

```
input  : $n$ (the number of terms left), $d$ (the desired total),
         $\{i_{\max}, j_{\max}, k_{\max}\}$
output: Returns "found" or "not found". If "found", the
         found value is output

$i_{\text{try}} \leftarrow 0$
while $i_{try} \leq 6$ do
    $j_{\text{try}} \leftarrow 0$
    while $j_{try} \leq 2$ do
        $k_{\text{try}} \leftarrow 0$
        while $k_{try} \leq 6$ do
            $t \leftarrow 2^{i_{\text{try}}} 3^{j_{\text{try}}} 5^{k_{\text{try}}} 79$
            if $t \leq 708$ then
                if multRecB(3, $711 - t$,
                $\{6 - i_{try}, 2 - j_{try}, 6 - k_{try}\}$) $=$ found then
                    Output $t$
                    return found
                end if
            end if
            $k_{\text{try}} \leftarrow k_{\text{try}} + 1$
        end while
        $j_{\text{try}} \leftarrow j_{\text{try}} + 1$
    end while
    $i_{\text{try}} \leftarrow i_{\text{try}} + 1$
end while
return not found
```

This approach terminates after trying 685 different choices for A, B, C, and D. If there were no solution, this approach would evaluate 746 total different choices.

2.1.5 OUR THIRD ADDITIVE SOLUTION

You could also attempt to combine the multiplicative and additive approaches more completely. Fundamentally, we are interested in stepping through every possible divisor of $711,000,000$ that is less than 711. This is effectively what the second additive approach is doing, but quite inefficiently. Approaching this more explicitly, we can simply enumerate each of the possible 61 divisors of $711,000,000$ less than 711, and then step through them, as in Algorithm 4 (or Listing 5 in Section 3).

Algorithm 4: Additive Brute Force Approach

```
output: Integer Values $A$, $B$, $C$, $D$ such that
        $ABCD = 711000000$ and $A + B + C + D = 711$

array divs[61]
= {1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, 18, 20, 24, 25, 30,
32, 36, 40, 45, 48, 50, 60, 64, 72, 75, 79, 80, 90, 96, 100,
120, 125, 144, 150, 158, 160, 180, 192, 200, 225, 237, 240,
250, 288, 300, 316, 320, 360, 375, 395, 400, 450, 474, 480,
500, 576, 600, 625, 632}

$i_A \leftarrow 1$
$A \leftarrow \text{divs}[i_A]$
while $i_A \leq 61$ do
    $i_B \leftarrow 1$
    $B \leftarrow \text{divs}[i_B]$
    while $A + B \leq 709$ and $i_B \leq i_A$ do
        $i_C \leftarrow 1$
        $C \leftarrow \text{divs}[i_C]$
        while $A + B + C \leq 710$ and $i_C \leq i_B$ do
            $D \leftarrow 711 - (A + B + C)$
            if $D \leq C$ then
                if $ABCD = 711000000$ then
                    return $A$, $B$, $C$, and $D$
                end if
            end if
            $i_C \leftarrow i_C + 1$
            $C \leftarrow \text{divs}[i_C]$
        end while
        $i_B \leftarrow i_B + 1$
        $B \leftarrow \text{divs}[i_B]$
    end while
    $i_A \leftarrow i_A + 1$
    $A \leftarrow \text{divs}[i_A]$
end while
```

This approach finds the result in 350 (much quicker) inner loops, and would perform at most $1,628$ inner loops if there were no solution.

Table 1 provides an approximate the run-time of each of these solutions.

Table 1: Runtime for Approaches

Implementation	Inner Loops	Clock Cycles	Normed Time
Additive I	808,883	68,294,499	170.64
Additive II	3,413	865,732	2.16
Additive III	350	400,225	1.00
Multiplicative I	9,844	9,006,862	22.50
Multiplicative II	685	537,159	1.34

2.2 MATHEMATICAL APPROACH

We could also attempt to solve this without a computer program, and exclude choices through mathematical trickery.

Looking at the factorization of $711000000 = 2^6 3^2 5^6 79^1$ we see that it has only one factor of 79, so only one of the item prices can have a factor of 79. Let's call this item price A.

$A \leq 708 = 711 - 3$, as no item is free (as the product is non-zero). This leaves us with the possible factorizations in Table 2.

Table 2: Options for A

Case	A	Added	Multiplied
1	$A = 2^0 3^0 5^0 79^1 = 79$	$B + C + D = 632$	$BCD = 9000000$
2	$A = 2^1 3^0 5^0 79^1 = 158$	$B + C + D = 553$	$BCD = 4500000$
3	$A = 2^2 3^0 5^0 79^1 = 316$	$B + C + D = 395$	$BCD = 2250000$
4	$A = 2^3 3^0 5^0 79^1 = 632$	$B + C + D = 79$	$BCD = 1125000$
5	$A = 2^0 3^1 5^0 79^1 = 237$	$B + C + D = 474$	$BCD = 3000000$
6	$A = 2^1 3^1 5^0 79^1 = 474$	$B + C + D = 237$	$BCD = 1500000$
7	$A = 2^0 3^0 5^1 79^1 = 395$	$B + C + D = 316$	$BCD = 1800000$

Some of these feel unlikely; let's firm that up.

2.2.1 CASES (4), (6) AND (7)

Cases (4), (6) and (7) are summarized in Table 3.

Table 3: Options for A (Cases (4), (6), and (7))

Case	A	Added	Multiplied
4	$A = 2^3 3^0 5^0 79^1 = 632$	$B + C + D = 79$	$BCD = 1125000$
6	$A = 2^1 3^1 5^0 79^1 = 474$	$B + C + D = 237$	$BCD = 1500000$
7	$A = 2^0 3^0 5^1 79^1 = 395$	$B + C + D = 316$	$BCD = 1800000$

The arithmetic/geometric inequality[2] tells us that:

$$\frac{1}{n}\sum_{i=1}^{n} x_i \geq \left(\prod_{i=1}^{n} x_i\right)^{1/n}$$

(with equality when all the values are equal). We can apply this inequality and massage it a bit, giving us:

$$\left(\frac{B+C+D}{3}\right)^3 \geq BCD$$

Plugging in our values, we see that this excludes (4), (6) and (7).

2.2.2 CASES (1), (2) AND (5)

Cases (1), (2) and (5) are summarized in Table 4

Table 4: Options for A (Cases (1), (2), and (5))

Case	A	Added	Multiplied
1	$A = 2^0 3^0 5^0 79^1 = 79$	$B + C + D = 632$	$BCD = 9000000$
2	$A = 2^1 3^0 5^0 79^1 = 158$	$B + C + D = 553$	$BCD = 4500000$
5	$A = 2^0 3^1 5^0 79^1 = 237$	$B + C + D = 474$	$BCD = 3000000$

In cases (1), (2) and (5), A is not divisible by 5, so BCD must be divisible by 5^6. At the same time, we see that $B+C+D$ is not divisible by 5, so at least one of the terms (say B) is not divisible by 5. This implies that CD must be divisible by 5^6.

Taking D to be the larger exponent of the 5 term (or possibly equal), we have D is divisible by 5^3, 5^4, 5^5 or 5^6. We can exclude 5^6 and 5^5 because they are larger than 711. D having a factor of 5^4 leaves at most $632 - 625 = 7$ for $B + C$, but C would be at least $5^2 = 25$ in this instance, so this is a contradiction. So, we are left with the case where both C and D are divisible by 125.

As such, we can describe B, C, and D as in Table 5 with the conditions in Table 6.

Table 5: Form of A, B, and C in Cases (1), (2), and (5)

$B = 2^{i_1} 3^{j_1}$	$C = 2^{i_2} 3^{j_2} 5^3$	$D = 2^{i_3} 3^{j_3} 5^3$

[2]`http://en.wikipedia.org/wiki/Inequality_of_arithmetic_and_geometric_means`

Table 6: Restrictions for Cases (1), (2), and (5)

Case	$i_1 + i_2 + i_3$	$j_1 + j_2 + j_3$	$B + C + D$
1	6	2	632
2	5	2	553
5	6	1	474

Note, neither j_2 or j_3 can equal 2, as that term would then exceed 632. If $j_3 = 1$, then $D = 3^1 5^3 = 375$, as any factor of 2 would make D too large. We then see that j_2 can't be 1, as $C + D$ would then exceed 632. We thus have two general cases to examine: the $j_3 = 1$ case, and the $j_2 = j_3 = 0$ case.

The $\mathbf{j_3 = 1}$ Case

Assume $j_3 = 1$, so $D = 3^1 5^3 = 375$. As previously mentioned, j_2 can't be 1.

So, in this case, we can describe B, C, and D as in Table 7, with the conditions in Table 6.

Table 7: Form of A, B, and C in Cases (1), (2), and (5) ($j_3 = 1$ Case)

$B = 2^{i_1} 3^{j_1}$	$C = 2^{i_2} 5^3$	$D = 3^1 5^3 = 375$

We now adopt the convention that B_m is the value for B in case (m).

Continuing, we know that $B_1 = B_2 = 2^{i_1} 3$ and $B_5 = 2^{i_1}$. $B + C \leq 257$, so we can immediately conclude that $C = 2^{i_2} 5^3$ has at most one factor of 2, as C would be greater than 257 otherwise. If C had exactly one factor of 2, then $C = 250$, leaving $B \leq 7$, a contradiction (as $B \geq 2^4 = 16$ in this instance). So, $C = 125$ and B has all the factors of two.

We have determined that in this case $D = 375$, and $C = 125$. In case (1) $B_1 = 2^6 3 = 192$, so $B_1 + C + D = 692 \neq 632$, a contradiction. In case (2) $B_2 = 2^5 3 = 96$, so $B_2 + C + D = 596 \neq 553$, a contradiction. In case (3) $B_3 = 2^6 = 64$, so $B_3 + C + D = 564 \neq 474$, a contradiction.

None of these cases works, so it can't be true that $j_3 = 1$.

The $j_2 = j_3 = 0$ Case

We now examine the case where $j_2 = j_3 = 0$. That is, B has all the 3 terms.

Table 8: Form of A, B, and C in Cases (1), (2), and (5) ($j_2 = j_3 = 0$ Case)

$B = 2^{i_1}3^{j_1}$	$C = 2^{i_2}5^3$	$D = 2^{i_3}5^3$

As C and D are now interchangeable, we can assume that C has the greater (or equal) power of 2.

First note that $i_2 \leq 2$, as greater powers make C too large.

Case $i_2 = 2$ (so $C = 500$): If $i_3 = 2$ (thus $D = 500$), then $C + D = 500 + 500 > 632$, a contradiction. If $i_3 = 1$ (thus $D = 250$), then $C + D = 500 + 250 > 632$, a contradiction. If $i_3 = 0$ (thus $D = 125$) then in cases (2) and (5) we have an immediate contradiction, as $C + D = 625$, which is already larger than the associated additive bounds (553 for case (2) and 474 for case (5)). For case (1) $B_1 = 2^4 3^2 = 144$, at which point $B_1 + C + D = 769 \neq 632$, which is a contradiction. None of these options worked, so $i_2 \neq 2$.

Case $i_2 = 1$ (so $C = 250$): If $i_3 = 1$ (so $D = 250$), then case (5) already runs into problems as $D + C = 500 > 474$, a contradiction. In case (2), $B_2 = 2^3 3^2 = 72$, so $B_2 + C + D = 572 \neq 553$, a contradiction. In case (1), $B_1 = 2^4 3^2 = 144$ whence $B_1 + C + D = 644 \neq 632$, a contradiction. If $i_3 = 0$ (so $D = 125$) we again get three cases. In case (1) $B_1 = 2^5 3^2 = 288$, then $B_1 + C + D = 663 \neq 632$. In case (2) $B_2 = 2^4 3^2 = 144$, whence $B_2 + C + D = 519 \neq 553$, a contradiction. In case (5) $B_5 = 2^5 3 = 96$, whence $B_5 + C + D = 471 \neq 474$. None of these worked, so $i_2 \neq 1$.

Case $i_2 = 0$ (so $C = 125$): This forces $D = 125$. In case (1), $B_1 = 2^6 3^2 = 576$, whence $B_1 + C + D = 826 \neq 632$. In case (2), $B_2 = 2^5 3^2 = 288$, whence $B_2 + C + D = 538 \neq 553$. In case (5), $B_5 = 2^6 3^1 = 192$, whence $B_5 + C + D = 442 \neq 474$. This also doesn't work.

As we have seen, none of these can occur, so we have excluded the cases (1), (2) and (5).

2.2.3 Case (3)

We are left with only case (3), so $A = 316$ and $B + C + D = 395$.

5^2 does not divide $B + C + D$, so we know that 5^2 doesn't divide at least one of the terms, say B. 5^4 is greater than 395, so C and D can have powers only up to 5^3. If both C and D had factors of 5^3, then $B = 395 - C - D$ would be divisible by 5, but this would then be a contradiction (as all the factors of 5 were consumed by the C and D terms). It must then be the case that B has a factor of 5^1, C has a factor of 5^2 and D has a factor of 5^3.

D cannot have a factor of 3 (if it did, it could only be a factor of 3^1 and then $D = 375$, $B + C = 20$ and $BC = 2^6 3^1 5^3$, so the arithmetic/geometric inequality would give us $B + C \geq 2^4\sqrt{375} > 20$, a contradiction.)

As such, A, B, and C must be of the form in Table 9 subject to the restrictions in Table 10.

Table 9: Form of A, B, and C in Case (3)

$B = 2^{i_1}3^{j_1}5$	$C = 2^{i_2}3^{j_2}5^2$	$D = 2^{i_3}5^3$

Table 10: Restrictions for Case (3)

Case	$i_1 + i_2 + i_3$	$j_1 + j_2$	$B + C + D$	BCD
3	4	2	395	2250000

We can immediately conclude that $i_3 \leq 1$, as any higher power of 2 results in $D > 395$, so D is either 125 or 250.

To proceed, we set up some new variables: Let $W = \frac{D}{125}$, $V = \frac{C}{25}$ and $U = \frac{B}{5}$. Plugging these into the sum and product equations in Table 10, we see that we have the equations:

$$\begin{aligned} 395 &= B + C + D \\ &= 5U + 25V + 125W \end{aligned}$$

or

$$79 = U + 5V + 25W$$

and

$$\begin{aligned} 2250000 &= BCD \\ &= (5U)(25V)(125W) \end{aligned}$$

or

$$144 = UVW$$

We have already established that either $W = 1$ or $W = 2$ (corresponding to $D = 125$ and $D = 250$, respectively).

If $W = 2$, then $29 = 5V + U$ and $UV = 72$ (and thus $U = \frac{72}{V}$). Substituting in, we arrive at $5V^2 - 29V + 72 = 0$, a quadratic polynomial. The quadratic equation tells us that this equation has only complex solutions, which is a contradiction (as V is complex, this certainly tells us that $C = 25V$ is complex, and we demanded integer solutions!)

Thus, $W = 1$ (so $D = 125$), which gives us $54 = 5V + U$ and $UV = 144$ (and thus $U = \frac{144}{V}$). Substituting in, we arrive at $5V^2 - 54V + 144 = 0$, again a quadratic polynomial. The quadratic equation gives us the solution $V = 6$ (thus $C = 150$). (We can discard the root associated with $V = \frac{24}{5}$, as this non-integer result would not yield a value for C which is divisible by 25). Plugging in, we find that $U = 24$, thus $B = 120$.

So, the solution is $A = 316$, $B = 120$, $C = 150$, $D = 125$ (in pennies), or $a = 3.16$, $b = 1.20$, $c = 1.50$, $d = 1.25$.

3 APPENDIX: CODE

Listing 1: Additive Brute Force Approach I

```
#include <stdio.h>

int main() {
  unsigned int A, B, C, D;

  for(A=1;A<=708;A++)
    for(B=1;A+B<=709 && B <= A;B++)
      for(C=1;A+B+C<=710 && C <= B;C++) {
        D = 711 - (A+B+C);
        if(D <= C) {
        if(A*B*C*D == 711000000) {
          printf(”A: %u, B: %u, C:%u, D: %u\n”,
                 A, B, C, D);
          return(1);
        }
      }
    }
  return(0);
}
```

Listing 2: Additive Brute Force Approach II

```
#include <stdio.h>

int main() {
  unsigned int A, B, C, D;

  for(A=1;A<=708;A++) {
    if(711000000 % A != 0) continue;
    for(B=1;A+B<=709 && B <= A; B++) {
      if(711000000 % B != 0) continue;
      for(C=1; A+B+C<=710 && C <= B; C++) {
        D = 711 - (A+B+C);
        if(D <= C) {
          if(A*B*C*D == 711000000) {
            printf(”A: %u, B: %u, C:%u, D: %u\n”,
                   A, B, C, D);
            return(1);
          }
        }
      }
    }
  }
  return(0);
}
```

Listing 3: Multiplicative Brute Force Approach I

```
#include <stdio.h>

/*Integer power function, returns base^(exp)
  Calculated using the standard square-and-multiply
  method of exponentiation*/
unsigned int ipow(unsigned int base,
                  unsigned int exp) {
  unsigned int result = 1;
  while (exp) {
    if (exp & 1)
      result *= base;
    exp >>= 1;
    base *= base;
  }

  return result;
}

/* Takes factors = {i, j, k} returns 2^i 3^j 5^k 79^l*/
unsigned int unfactor(unsigned int *factors) {
  return(ipow(2, factors[0]) * ipow(3, factors[1]) *
      ipow(5, factors[2]) * ipow(79,factors[3]));
}

int tryVar(unsigned int termsLeft, unsigned int
    desiredTotal, unsigned int *fList) {
  unsigned int curTry[4];
  unsigned int nextRound[4];
  unsigned int total;
  int i;

  if(termsLeft == 1) {
    total = unfactor(fList);
    if(total == desiredTotal) {
      printf(”n = 1, %u\n”, total);
      return(1);
    }
  } else {
    for(curTry[0]=0; curTry[0]<=fList[0]; curTry[0]++)
      for(curTry[1]=0; curTry[1]<=fList[1];
          curTry[1]++)
        for(curTry[2]=0; curTry[2]<=fList[2];
            curTry[2]++)
          for(curTry[3]=0; curTry[3]<=fList[3];
              curTry[3]++) {
            total = unfactor(curTry);
            if(total <= desiredTotal - termsLeft + 1) {
              for(i=0; i<4; i++)
                 nextRound[i]=fList[i]-curTry[i];
              if(tryVar(termsLeft-1,
                 desiredTotal-total,
                 nextRound)) {
```

```
                printf(”n = %u, %u\n”, termsLeft,
                       total);
                return(1);
              }
            }
          }
  }

  return(0);
}

int main()
{
  unsigned int fList[4] = {6, 2, 6, 1};

  tryVar(4, 711, fList);

  return 0;
}
```

Listing 4: Multiplicative Brute Force Approach II

```
#include <stdio.h>

/*Integer power function, returns base^(exp)
  Calculated using the standard square-and-multiply
  method of exponentiation*/
unsigned int ipow(unsigned int base,
                  unsigned int exp) {
  unsigned int result = 1;
  while (exp) {
    if (exp & 1)
      result *= base;
    exp >>= 1;
    base *= base;
  }

  return result;
}

/* Takes factors = {i, j, k} returns 2^i 3^j 5^k */
unsigned int unfactor(unsigned int *factors) {
  return(ipow(2, factors[0]) * ipow(3, factors[1]) *
      ipow(5, factors[2]));
}

int tryVar(unsigned int termsLeft,
           unsigned int desiredTotal,
           unsigned int *fList) {
  unsigned int curTry[3];
  unsigned int nextRound[3];
  unsigned int total;
  int i;

  if(termsLeft == 1) {
    total = unfactor(fList);
    if(total == desiredTotal) {
      printf(”n = 1, %u\n”, total);
      return(1);
    }
  } else {
    if(ipow(desiredTotal,termsLeft) >= unfactor(fList)*
       ipow(termsLeft, termsLeft)) { //note (1)
      for(curTry[0]=0; curTry[0]<=fList[0];
          curTry[0]++)
        for(curTry[1]=0; curTry[1]<=fList[1];
            curTry[1]++)
         for(curTry[2]=0; curTry[2]<=fList[2];
             curTry[2]++) {
            total = unfactor(curTry);
            if(total <= desiredTotal - termsLeft + 1) {
              for(i=0; i<3; i++)
                nextRound[i]=fList[i]-curTry[i];
              if(tryVar(termsLeft-1,
                      desiredTotal-total, nextRound))
                          {
                printf(”n = %u, %u\n”, termsLeft,
                      total);
                return(1);
              }
            }
          }
      }
    }

    return(0);
  }

  int main()
  {
    unsigned int fList[3] = {6, 2, 6};
    unsigned int curTry[3];
    unsigned int nextRound[3];
    unsigned int total;
    int i;

    for(curTry[0]=0; curTry[0]<=fList[0]; curTry[0]++)
      for(curTry[1]=0; curTry[1]<=fList[1]; curTry[1]++)
       for(curTry[2]=0; curTry[2]<=fList[2];
           curTry[2]++) {
         total = 79 * unfactor(curTry);
         if(total <= 708) {
           for(i=0; i<3; i++)
               nextRound[i]=fList[i]-curTry[i];
           if(tryVar(3, 711-total, nextRound)) {
             printf(”n = %u, %u\n”, 4, total);
             return(1);
           }
         }
       }

    return(0);
  }
```

Listing 5: Additive Brute Force Approach III

```
int main() {
  unsigned int A, B, C, D;
  unsigned int Aind, Bind, Cind;
  unsigned int divs[61] =
    {1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, 18, 20,
    24, 25, 30, 32, 36, 40, 45, 48, 50, 60, 64, 72,
    75, 79, 80, 90, 96, 100, 120, 125, 144, 150,
    158, 160, 180, 192, 200, 225, 237, 240, 250,
    288, 300, 316, 320, 360, 375, 395, 400, 450,
    474, 480, 500, 576, 600, 625, 632};

  for(Aind=0, A=divs[0]; Aind<61; Aind++, A=divs[Aind])
    for(Bind=0, B=divs[0]; A+B<=709 && Bind<=Aind;
        Bind++, B=divs[Bind])
      for(Cind=0, C=divs[0]; A+B+C<=710 && Cind<=Bind;
          Cind++, C=divs[Cind]) {
        D = 711 - (A+B+C);
        if(D<=C) {
          if(A*B*C*D == 711000000) {
            printf(”A: %u, B: %u, C:%u, D: %u\n”,
                   A, B, C, D);
            return(1);
          }
        }
      }

  return(0);
}
```

Colophon

The text of this document is typeset in Jean-François Porchez's wonderful Sabon Next typeface. Sabon Next is a modern (2002) revival of Jan Tschichold's 1967 Sabon typeface, which is in turn a adaptation of the classical (in all meanings) Garamond typeface, which dates from the early 16th century.

Equations are typeset using the MathTime Professional II (MTPro2) fonts, a font package released in 2006 by the great mathematical expositor Michael Spivak. These fonts are designed to work with the Times typeface, but they blend well with most classical fonts.

Source listings are typeset in Microsoft's Consolas, a monospace font with excellent readability.

XeTeX was used to typeset the document, which is in turn offspring of Donald Knuth's profoundly important TeX. XeTeX was selected in order to gain access to modern fonts without the trauma involved in converting them to a representation that pdfTeX could deal with. This approach makes most (though sadly, not all) OpenType features available, and sidesteps the traditional limit of 256 glyphs per font.