THE 711 PROBLEM[*]

David Gries

TR 82-493
May 1982

Department of Computer Science
Upson Hall
Cornell University
Ithaca, New York 14853

*This work was supported by NSF grant MCS81-03605.

The 711 Problem

David Gries
Computer Science Department
Cornell University

__Introduction__. The United States is filled with small grocery stores that, for convenience, are open at all hours of the day and night. In the south, these are called 711 stores, because originally they were open from seven in the morning until eleven at night.

One day, a customer bought 4 items at a 711 store. The cashier bagged them and said, "that will be \$7.11, please." The customer asked, "Is it \$7.11 because this is a 711 store?" "No", replied the cashier, "I multiplied the prices together and got \$7.11." "But you're supposed __add__ them, not multiply them", said the customer. "Oh, you're right!" exclaimed the cashier. "Let me recalculate ... that will be \$7.11."

What were the prices of the 4 items?

From this puzzle we can extract the following problem. Write a program that, given two positive integers N and M, will find 4 integers, called b, c, d, e, satisfying

(1) $$0 \le b, c, d, e \le N,$$
$$b+c+d+e = N,$$
$$b*c*d*e = M.$$

Call such a tuple (b, c, d, e) a __solution__. If no solution exists, indicate that in some fashion.

For the original puzzle, use $N = 711$ and $M = 711000000$, and if (b, c, d, e) is a solution the prices are $b/100$, $c/100$, $d/100$ and $e/100$.

M is likely to be much larger than N, as is the case in the 711 problem, so an $O(M)$ algorithm is not as good as an $O(N^2)$ or even an $O(N^3)$ algorithm. So let us put aside possible algorithms that deal with finding prime factors of M. Instead, think of searching the space of 4-tuples (b, c, d, e) for a solution. The problem is to organize the search efficiently. We give two $O(N^2)$ (in the worst case) solutions.

__The First Algorithm__. Consider a fixed pair $\bar{b}, \bar{c}$. If $(\bar{b}, \bar{c}, d, e)$ is a solution, we have

$$\bar{b}+\bar{c}+d+e = N \quad \text{and} \quad \bar{b}*\bar{c}*d*e = M.$$

Solving the first equation for d, substituting for d in the second, and rearranging leads to

(2) $\bar{b}*\bar{c}*e^2 - \bar{b}*\bar{c}*(N-\bar{b}-\bar{c})*e + M = 0$.

If this equation has an integer solution e satisfying $0 \le e \le N$ and $0 \le N-\bar{b}-\bar{c}-e \le N$, then $(\bar{b}, \bar{c}, N-\bar{b}-\bar{c}-e, e)$ is a solution to the problem.

It is now easy to write an algorithm that searches through pairs (b, c), looking for a solution e to (2). However, this algorithm will involve finding a square root of a rather large integer at each step, so let us look for a better algorithm.

Towards a second algorithm. Any solution can be arranged in the form (b, c, d, e) where $b \le c \le d \le e$, so consider searching only the space of lexically ordered 4-tuples. At any stage, four variables b, c, d, e will contain integers, and it will be known that no 4-tuple with first component $< b$ is a solution. The problem, given fixed b, is to search the space of ordered tuples (c, d, e) so that not all ordered tuples need be tested.

Notice that one of the components in a solution, say e, is completely determined by the other three: $e = N-(b+c+d)$. Thus, for fixed b we need search only through pairs (c, d). To find a way of reducing the order of execution even further, let us look at Saddleback Search.

Saddleback Search [1, page 215]. Given is an array $f[0{:}m, 0{:}n]$. Each row and each column is ordered (in ascending order). A value x lies in f; the problem is to determine its row and column number (if x is in f more than once, find any one of its positions). The following neat algorithm solves the problem in time linear in the number of rows plus the number of columns in the worst case, and this is about the best one can do. The algorithm begins by identifying a rectangular section of f in which x appears, then iteratively reduces the size of this rectangle, always maintaining the fact that x lies in it. In another sense, it begins looking for x at the upper right element $f[0,n]$ and proceeds towards the lower left element $f[m,0]$; the fact that the rows and columns are ordered allows the search to proceed efficiently.

```
i,j:= 0,n;
{inv: $0 \le i \le m \wedge 0 \le j \le n \wedge x \in f[i{:}m, 0{:}j]$}
{bound function: $m-i+j$}
do f[i,j] > x → j:= j-1
 [] f[i,j] < x → i:= i+1
od
{x = f[i,j]}
```

Using Saddleback-Search in the 711 Problem. In the abstract 711 problem, for fixed b we must search pairs (c, d) satisfying $0 \le b \le c \le d \le N-b-c-d$ for one that satisfies a certain property. Consider a two-dimensional array f[b:N,b:N]. The value f[c,d] is b*c*d*(N-b-c-d), the product of the components of 4-tuple (b, c, d, N-b-c-d). If each row and column of f is ordered, we can use Saddleback Search to determine whether M is in f. This would reduce the time to search the 4-tuples (b, c, d, N-b-c-d) for fixed b to O(N), thus reducing the time for the complete algorithm to $O(N^2)$. Further, since each array element is a function of its subscripts, there is no need to maintain the array itself.

Array f does not have the desired property (rows and columns ordered), but it is close enough so that a Saddleback-like search can still be used.

The Second Algorithm. A discussion follows the algorithm.

```
b, c, d, e, m:= 0, 0, 0, N, 0;
{invariant: P0}
{bound function: N-4*b}
do m ≠ M ∧ 4*(b+1) ≤ N →
    b:= b+1;
    c, d, e:= b, floor((N-2*b)/2), ceil((N-2*b)/2);
    m:= b*c*d*e;
    {inv: P}
    {bound function:  d-c}
    do m ≠ M  ∧  c < d
        L: if m > M  ∨  d = e → d, e:= d-1, e+1;  m:= b*c*d*e
            ▯ m < M  ∧  d < e → c, e:= c+1, e-1;  m:= b*c*d*e
           fi
    od
    {P0 ∧ (m = M ∨ c = d)}
od
```

The algorithm begins with (b, c, d, e) = (0, 0, 0, N). Each iteration of the main loop of the algorithm increases b by 1 and searches the 4-tuples with first component b for a solution. A variable m is used to contain the value b*c*d*e. Thus, the invariant of the main loop is

P0: $0 \le b \wedge b*c*d*e = m \wedge$
no ordered solution with first component $\le b$ exists, unless (b, c, d, e) is an ordered solution

A bound function of the main loop is N-4*b. If the body of the main loop maintains P0, it is clear that the algorithm is correct.

The inner loop searches for a solution among all 4-tuples with first component b. It maintains $c \le d \le e = N-b-c-d$; further, any ordered solution $(b, \bar{c}, \bar{d}, N-b-\bar{c}-\bar{d})$ satisfies $c \le \bar{c} \le \bar{d} \le d$. Each iteration increases c or decreases d, at the same time changing e to maintain $b+c+d+e = N$. The invariant of the inner loop is therefore

P: $0 \le b \le c \le d \le e \le N$ $\wedge$
$b+c+d+e = N$ $\wedge$ $b*c*d*e = m$ $\wedge$
no solution with first component $< b$ exists $\wedge$
any ordered solution $(b, \bar{c}, \bar{d}, N-b-\bar{c}-\bar{d})$ satisfies $c \le \bar{c} \le \bar{d} \le d$

We now investigate the correctness of the body L of the inner loop, assuming that the loop guard is true. L must satisfy

$$m \ne M \wedge c < d \wedge P \Rightarrow wp(L,P)$$

Clearly, execution of L leaves the first three lines of P true. Hence we need only prove the invariance of the last conjunct of P. Let us consider each of the guarded commands of the alternative command, in turn.

Because execution of the first guarded command reduces d, it must be shown that no ordered solution exists with first component b and third component d. Suppose $m > M$. Below, we list all 4-tuples with the current values of b and d that satisfy $b+c+d+e = N$, in increasing lexical order:

(0) $(b,0, d,e+c)$
(1) . . .
(2) $(b,c-1,d,e+1)$
(3) (b,c, d,e)
(4) $(b,c+1,d,e-1)$
(5) . . .
(6) $(b,c+j,d,e-j)$ (either $c+j = d$ or $e-j = d$)
(7) $(b,c+j+1,d,e-j-1)$
(10) . . .

The 4-tuples given by (0)-(2) have second component less than c, and invariant P indicates that these cannot be ordered solutions. The products of the four components of the 4-tuples (3)-(6) are in increasing order (since $c < e$), and since $m = b*c*d*e > M$, none of these tuples can be solutions. Finally, each of the tuples (7-11) is not lexically ordered, since either $c+j = d$ or $d = e-j$. Hence, none of the tuples (0)-(10) are ordered solutions and no ordered solution with first component b and third component d exists.

Now suppose that $d = e$. We again investigate 4-tuples with first component b and third component d, in increasing lexical order:

```
(0) (b,0,  d,d+c)
(1)  . . .
(2) (b,c-1,d,d+1)
(3) (b,c,  d,d  )
(4) (b,c+1,d,d-1)
(5)  . . .
```

By invariant P, tuples (0)-(2) cannot be ordered solutions. Because the guard of the main loop is true, the product of the components of tuple (3) is not M, so tuple (3) is not a solution. Further, tuples (4)-(5) are not ordered, so they are not ordered solutions. Hence, no ordered solution exists with first component b and third component d.

Now consider the second guarded command. For it to maintain the last conjunct of P, no ordered 4-tuple with first two components b and c can be a solution. We leave it to the reader to show this by examining the list of lexically ordered 4-tuples with first components b and c under the assumption that the guard $m < M \wedge d < e$ is true.

Thus, under all cases execution of L leaves P true. Since

$$m \neq M \wedge c < d \wedge P$$

implies that at least one of the guards of L is true, abortion cannot occur. Hence execution of the loop body performs the desired task.

The inner loop terminates with

$$\{P \wedge (m = M \vee c \geq d)\}$$

true. If $m = M$ then a solution has been found; if not, $c = d$ and, with the help of invariant P, we see that no ordered solution with first component b exists. Therefore, execution of the body of the main loop leaves PO true, and the algorithm is correct.

This algorithm was translated into Pascal and executed on a VAX. For the initial 711 problem, the single solution ($1.20, $1.25, $1.50, $3.16) was determined.

Acknowledgements. I am grateful to Don Edwards of USNSW, Dahlgren, Virginia, for bringing the problem to my attention. The first algorithm is due to Jay Misra.

References.
[1] Gries, D. The Science of Programming. Springer Verlag, New York, 1981.