The Reve's Puzzle

J. S. ROHL* and T. D. GEDEON
Department of Computer Science, University of Western Australia, Nedlands 6009, Western Australia

The Towers of Hanoi problem, explored in many recent Journal papers, is extended to consider a related problem, The Reve's Puzzle. It is shown that The Reve's Puzzle is a generalisation of the Towers of Hanoi problem and a number of questions concerning the elegance of the algorithms are posed.

Received January 1985

The Towers of Hanoi,[4] a puzzle now more than 100 years old, has had a new lease of life in the computer age, with a particular resurgence in the last couple of years when a number of papers have been published, in this journal particularly.

We would like to present a related problem, the Reve's Puzzle introduced in 1907 by Dudeney,[1] who has been described by Martin Gardner[2] as England's Greatest Puzzlist. The puzzle is described in *The Canterbury Puzzles*, whose format is based on Chaucer's *Canterbury Tales*. The description goes as follows.

When the pilgrims were stopping at a wayside tavern, a number of cheeses of varying sizes caught his alert eye; and calling for four stools, he told the company that he would show them a puzzle of his own that would keep them amused during their rest. He then placed eight cheeses of graduating sizes on one of the end stools, the smallest cheese being at the top (and no cheese resting on a smaller one.) 'This is the riddle', quoth he, 'that I did once set before my fellow townsmen at Baldeswell, that is in Norfolk, and, by Saint Joce, there was no man among them that could rede it aright. And yet it is withal full easy, for all that I do desire is that, by the moving of one cheese at a time from one stool unto another, ye shall remove all the cheeses to the stool at the other end without ever putting any cheese on one that is smaller than itself. To him that will perform this feat in the least number of moves that be possible will I give a draught of the best that our good host can provide.'

Clearly, this is a generalisation of the Towers of Hanoi problem, and a number of authors, including the present ones,[3] have described it as such. However, in view of Dudeney's delightful setting and because he also solved the problem it seems fitting to use his name for it.

To set the scene we give first of all a version of the Hanoi procedure tailored to cheeses and stools. It assumes the types:

```
ncheeses = 0..maxcheeses;
stools = 1..maxstools
```

where *maxcheeses* is defined appropriately, and *maxstools* is, at this point, 3.

```
procedure Hanoi (n:ncheeses; s1, s2, s3: stools);
begin
if n = 1 then
writeln ('Move a cheese from stool', s1:1, 'to stool', s2:1)
else
begin
Hanoi (n-1, s1, s3, s2);
writeln ('Move a cheese from stool', s1:1, 'to stool', s2:1);
Hanoi (n-1, s3, s2, s1)
end
end; {of procedure 'Hanoi'}
```

* To whom correspondence should be addressed.

Of course Dudeney gave a solution to the problem. It is expressed below with reference to 10 cheeses (not the 8 of the question as posed).

we first pile the six smallest cheeses in 17 moves on one stool; then we pile the next 3 cheeses on another stool in 7 moves; then remove the largest cheese in 1 move; then replace the 3 in 7 moves; and finally replace the 6 in 17: making in all the necessary 49 moves.

In this description, the middle three steps are a clear description of the solution of the Towers of Hanoi problem with 3 cheeses. Thus the solution to the Reve's problem can be expessed in terms of *Hanoi*. If the number of cheeses moves by *Hanoi* is called *F3*(*n*), then we have:

```
procedure Reve (n:ncheeses; s1, s2, s3, s4: stools);
begin
if n = 1 then
writeln ('Move a cheese from stool', s1:1, 'to stool', s2:1)
else
begin
Reve (n-F3(n), s1, s4, s3, s2);
Hanoi (F3(n), s1, s2, s3);
Reve(n-F3(n), s4, s2, s3, s1)
end
end; {of procedure 'Reve'}
```

But what is this function *F3*? Dudeney knew that too! He gave a table showing that if *n* were a triangular number:

1 3 6 10 . . . $i(i+1)/2$ . . .

then *F3*(*n*) is the ordinal of the triangular number:

1 2 3 4 . . . i . . .

He left as an exercise to the reader the case where *n* is not a triangular number. In fact it is the ordinal of the smallest triangular number not less than *n*. Thus the function *F3* is:

```
function F3 (n:ncheeses):ncheeses;
begin
F3:= trunc((sqrt(1+8*n)-1)/2)
end; {of function 'F3}'
```

Now *Reve* not only calls *Hanoi*, it is a generalisation of it. This can best be seen if we introduce an even simpler puzzle, where there are only two stools. It is impossible to move more than 1 cheese, but in the light of what follows we retain the parameter *n*.

```
procedure Simple (n: ncheeses; s1, s2:stools);
begin
writeln ('Move a cheese from stool', s1:1, 'to stool', s2:1)
end; {of procedure 'Simple'}
```

If we introduce a function *F2*, then we can write *Hanoi* in terms of *Simple*.

```
procedure Hanoi (n:ncheeses; s1, s2, s3:stools);
begin
if n = 1 then
writeln ('Move a cheese from stool', s1:1, 'to stool', s2:1)
else
begin
Hanoi(n-F2(n), s1, s3, s2);
Simple (F2(n), s1, s2);
Hanoi(n-F2(n), s3, s2, s1)
end
end; {of procedure 'Hanoi'}
```

F2(*n*) is, of course, 1.

The similarity between *Hanoi* and *Reve* is now manifest and we can easily write a procedure which includes them both, and *Simple* as well. We need an extra parameter, *m* of type *nstools* = 2..*maxstools*, which specifies the number of stools; the stools are represented by an array; and the functions *F2* and *F3* are merged into a function *F*, which takes a further parameter giving the number of stools.

```
procedure Reve (m:nstools; n:ncheeses; s:stools);
begin
if n = 1 then
writeln ('Move a cheese from stool', s[1]:1, 'to stool',
    s[2]:1)
else
begin
swap(s[2], s[m]);
Reve(m, n-F(m-1, n), s);
swap (s[2], s[m]);
Reve(m-1, F(m-1, n), s);
swap(s[1]), s[m]);
Reve(m, n-F(m-1, n), s);
swap(s[1], s[m])
end
end; {of procedure 'Reve'}
```

Note that this procedure has two recursive aspects. At any call, one or both of the parameters *m* and *n* is reduced, the recursion stopping when *n* = 1. (When *m* becomes 2, *n* always becomes 1.) Teachers may find this a useful alternative example to Ackermann's function in this respect.

This new procedure works for a larger number of stools, as well. All we need is to increase the number of stools and determine the appropriate function *F*. Once again, Dudeney had the answer. He noted that for five stools the pertinent numbers were the pyramidal numbers:

1, 4, 10, 20, . . ., $(i(i+1)(i+2)/6$

rather than the triangular numbers.

Enter Pascal's Triangle:

```
1
1  1
1  2   1
1  3   3   1
1  4   6   4   1
1  5  10  10   5   1
1  6  15  20  15   6   1
1  7  21  35  35  21   7   1
```

To find *F*(*m*, *n*), move down the *m*th column to the largest number not greater than *n*. If this is the *k*th element of the column, then *F*(*m*,*n*) is the *k*th element of the column to its left.

The problem has been considered in the recreational mathematics literature,[5] where the main interest has been in finding the minimum number of moves required. Our algorithm is optimal in that it generates the minimum number of moves. In computer science we are interested in generating those moves. A number of questions arise:

- We have not given the function *F*. Is there an elegant algorithm which avoids recalculating some values of *F*(*m*,*n*)?
- Our algorithm requires elements of the array to be swapped. Is there an elegant form of the algorithm that avoids this?
- Does there exist an elegant algorithm in which one of the recursive aspects is replaced by iteration?
- Ditto for both recursive aspects.

REFERENCES

1. H. E. Dudeney, *The Canterbury Puzzles*, Thomas Nelson & Son, London (1907) (4th Edition of 1919 reprinted, and published by Dover in 1958).
2. Martin Gardner, *More Mathematical Puzzles and Diversions*, Penguin, London (1966).
3. J. S. Rohl & T. D. Gedeon, Four-Tower Hanoi and beyond, *Proceedings of 6th Australian Computer Science Conference, Sydney* (1983).
4. W. W. Rouse Ball, *Mathematical Recreations and Essays*, Macmillan, London (1892).
5. B. M. Stewart, Problem 3918 and a Solution, *American Mathematical Monthly* **46**, 363 (1939) and **48**, 216–219 (1941).