Fibonacci Numbers Using Mutual Recursion

Manuel Rubio
Rey Juan Carlos University
Department of Computer Science, Statistics and Telematics
c/ Tulipán s/n, 28-933 Madrid, Spain
manuel.rubio@urjc.es

Bozena Pajak
Jagiellonian University
School of Philosophy
Department of Sociology
ul. Grodzka 52, 31-044 Krakow, Poland
bozenapajak@interia.pl

ABSTRACT

The calculation of Fibonacci numbers is a very popular programming problem. It has several iterative and recursive solutions that are often analyzed in programming courses. This paper proposes an alternative algorithm based on mutual recursion that is derived from Fibonacci's famous rabbits experiment. The main contribution of our approach is that it enables students to view the problem from a different perspective. This can motivate them to apply the functional paradigm to everyday problems by focusing on recursive strategies and the concept of induction, rather than on the underlying mathematical functions being implemented.

1. INTRODUCTION

The calculation of Fibonacci numbers is a classical programming problem that has been studied in depth by many computer scientists and mathematicians [2, 8, 6]. Formally, the Fibonacci number F_n is the n-th term of the series formed by the following recurrence relation:

$$F_n = F_{n-1} + F_{n-2} \tag{1}$$

for $n = 3, 4, \ldots$, with $F_1 = F_2 = 1$ (some references also consider the term $F_0 = 0$). Computer scientists find this problem highly interesting from a pedagogical point of view. This is mainly because of its simplicity, which permits several intuitive solutions based on different programming paradigms (e.g. imperative, functional). It is therefore often discussed in programming courses as a means of explaining and comparing different programming strategies. In particular, it is analyzed in introductions to multiple recursion along with other problems, such as the Towers of Hanoi [3, 1] or the calculation of binomial coefficients [9].

The closed-form Fibonacci number formula is one of the approaches for calculating F_n:

$$F_n = \frac{\phi^n - (1-\phi)^n}{\sqrt{5}}$$

where $\phi = \frac{1+\sqrt{5}}{2} \approx 1.618$ is the golden ratio [10]. However, it is hardly implemented in practice due to the fact that the floating point powers are slow and suffer from rounding errors.

A popular and efficient solution consists of using an iterative algorithm where a simple loop and a few variables are sufficient to compute successive Fibonacci numbers until reaching F_n. Another iterative solution uses an array to store the first n Fibonacci numbers. Both of these approaches have $\theta(n)$ time complexity. Moreover, the process can be accelerated (down to $\theta(\log n)$ time complexity) using exponentiation by squaring for high values of n [4].

```
int Fib_Rec(int n)
{
   if((n==1)||(n==2))
      return 1;
   else
      return Fib_Rec(n-1) + Fib_Rec(n-2);
}
```

Figure 1: Classical Fibonacci recursive algorithm.

There are also many recursive solutions for calculating Fibonacci numbers. The most popular approach discussed with students is the algorithm shown in Figure 1. It may be considered more intuitive than iterative versions since it can be derived directly from the definition. It therefore constitutes a powerful straightforward example of (multiple) recursion. Its main disadvantage, however, is its exponential time complexity ($\theta(\phi^n)$) [9]. Even though some programming languages can accelerate its computation by using memoization, it is seldom used for non-academic purposes. On the other hand, there exist several sophisticated recursive strategies that make use of Fibonacci identities [5] in order to lower the time complexity to $\theta(n)$. Tabulation techniques may also be applied in order to reduce the complexity even further to $\mathcal{O}(\log n)$ [8].

Despite the ostensible simplicity of the `Fib_Rec` function, the interpretation of F_n by examining its recursion tree is not trivial, especially for high values of n. From a mathematical point of view it is clear that F_n is equal to the number of function calls to `Fib_Rec(2)` and `Fib_Rec(1)` (leaves of the tree). `Fib_Rec(2)` is called F_{n-1} times and `Fib_Rec(1)` is called F_{n-2} times, which sum F_n in the general case ($n > 2$). Note that `Fib_Rec(i)` is called F_{n-i+1} times, except for `Fib_Rec(1)`, which is called F_{n-2} times ($n > 2$). However, counting the number of leaves is rather complicated because of the recursion tree's structure. Its leaves are not always located on the same level since it is a pruned binary tree. Figure 2 shows the recursion tree for `Fib_Rec(8)`, where there are $F_8 = 21$ shaded leaves spread out over four different levels. On the other hand, the recursion tree adequately reflects the recurrence relation in (1).

This paper proposes an alternative recursive algorithm related to Fibonacci's famous rabbits experiment, from which the Fibonacci sequence (1,1,2,3,5,8,13,21,...) was first derived. Our approach offers a new solution based on the con-

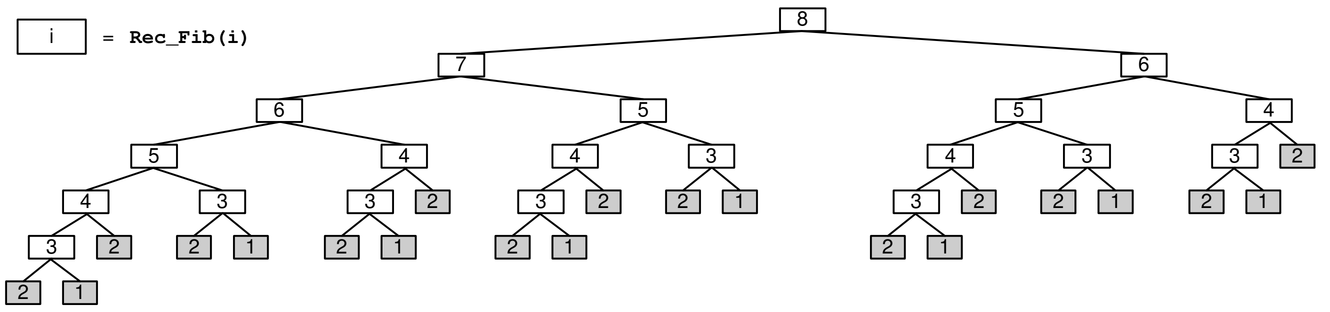


Figure 2: Recursion tree for `Fib_Rec(8)`.

Month	Baby pairs	Adult pairs	Total pairs
1	1	0	1
2	0	1	1
3	1	1	2
4	1	2	3
5	2	3	5
6	3	5	8
7	5	8	13
8	8	13	21
⋮	⋮	⋮	⋮

Table 1: Rabbits population growth process.

cept of mutual recursion, where the leaves of the recursion tree are all located on the same level, and can also add up to F_n.

The rest of the paper is organized as follows: Section 2 describes Fibonacci's experiment. Our proposed algorithm and several variants are discussed in Section 3. The last section presents a discussion and our final conclusions.

2. FIBONACCI'S RABBITS PROBLEM

The origin of the Fibonacci numbers and their sequence can be found in a thought experiment posed in 1202. It was related to the population growth of pairs of rabbits under ideal conditions. The objective was to calculate the size of a population of rabbits during each month according to the following rules (the original experiment was designed to calculate the population after a year):

1. Originally, a newly-born pair of rabbits, one male and one female, are put in a field.
2. Rabbits take one month to mature.
3. It takes one month for mature rabbits to mate and give birth to another pair of newly-born rabbits.
4. The female always gives birth to one male and one female rabbit every month from the second month on.
5. Rabbits never die.

Table 1 describes the population growth process with a linear structure where baby and adult pairs are treated separately. This distinction is fundamental in order to derive our proposed algorithm in Section 3.

On one hand, it is easy to see that the sequences of baby, adult and total number of pairs of rabbits follow the recurrence relation given in (1). The difference between them is caused by their first two elements: 1 and 0 for the babies, 0 and 1 for adults, and 1 and 1 for the total number of pairs of rabbits. Thus, this last sequence is the Fibonacci sequence.

On the other hand, let B_i, A_i and F_i be the number of baby, adult and total number of pairs of rabbits, respectively, at month i. We then have the following trivial identities:

$$B_i = \begin{cases} 1 & i = 1 \\ A_{i-1} & i \neq 1 \end{cases} \tag{2}$$

Since every adult pair mates and gives birth to a new pair of babies every month, the number of baby pairs at month i is equal to the number of adult pairs in the previous month $i-1$.

$$A_i = \begin{cases} 0 & i = 1 \\ A_{i-1} + B_{i-1} & i \neq 1 \end{cases} \tag{3}$$

The number of adult pairs at month i is the sum of the adult pairs in the previos month $i-1$ (since rabbits never die), plus the number of baby pairs that will have matured over the previous month.

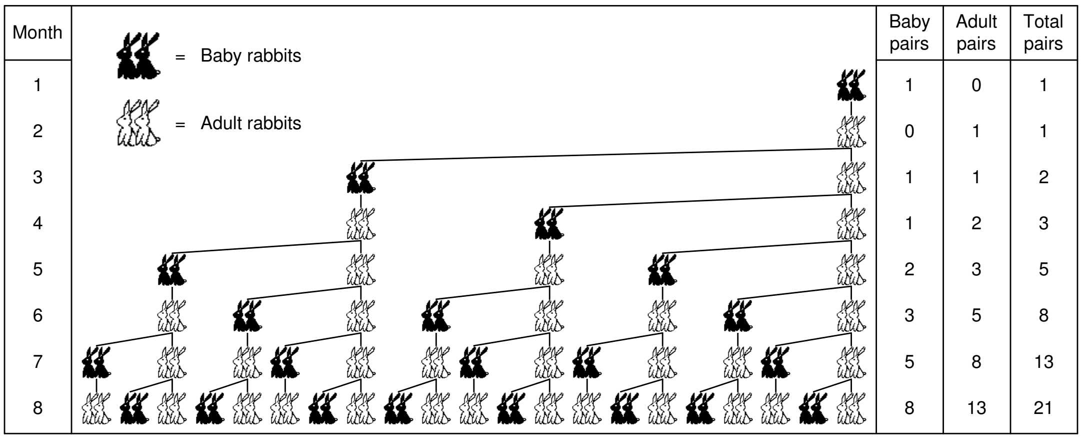
$$F_i = A_i + B_i = A_{i+1} = B_{i+2} \tag{4}$$

Other similar identities can also be easily inferred [5].

A linear table is an unorthodox way of explaining the rabbit population growth process. Most print and web references describe it with a pseudo-genealogical tree similar to the one shown in Figure 3, where generally no distinction is made between baby pairs and adult pairs (this difference turns out to be crucial for the development of our algorithm). The tree is not a true genealogical tree, since in a particular month (tree level), all pairs of rabbits are shown. In other words, the same pair of rabbits may appear several times on the tree (in this case, there is a one-to-one relationship between a specific pair of rabbits and the column it appears in). An analysis of the tree's structure reveals that the total number of rabbit pairs alive at month n is equal to F_n, and therefore to its number of leaves, which all appear on the same level (note that this was not the case in the recursion tree shown in Figure 2). The pseudo-genealogical tree is interesting in this study because it basically has the same structure as the recursion tree (see Figure 5) associated with our proposed algorithm. Nevertheless, our experience indicates that the rabbit population growth process is easier to comprehend with a linear table.

3. PROPOSED ALGORITHM

In this section we will show that a natural solution to the Fibonacci's rabbits problem provides a different, simple, and



Month	Baby pairs	Adult pairs	Total pairs
1	1	0	1
2	0	1	1
3	1	1	2
4	1	2	3
5	2	3	5
6	3	5	8
7	5	8	13
8	8	13	21

Figure 3: Pseudo-genealogical tree.

```
int Babies(int i)
{
   if(i==1)
     return 1;
   else
     return Adults(i-1);
}

int Adults(int i)
{
   if(i==1)
     return 0;
   else
     return Adults(i-1) + Babies(i-1);
}

int Fib_Mutual_Rec(int n)
{
   return Adults(n) + Babies(n);
// return Adults(n+1); // is also valid
// return Babies(n+2); // is also valid
}
```

Figure 4: Proposed Fibonacci mutual-recursive algorithm.

pleasing example of two mutually recursive functions. While the classical recursive Fibonacci algorithm is based on (1), our version is based on (2), (3) and (4). The implementation of the recursive functions is straightforward, as shown in Figure 4. Note that in order to match the algorithm's recursion tree (see Figure 5) to the pseudo-genealogical tree in Figure 3 `Fib_Mutual_Rec(n)` must call `Babies(n+2)`.

3.1 Variants

In the previous setting the calculated Fibonacci number is equal to the number of calls to `Babies(1)` (shaded leaves on the recursion tree). Slight modifications to the functions can generate algorithms that instead return the number of calls to the unshaded leaves (`Adults(1)`):

$$B'_i = \begin{cases} 0 & i=1 \\ A'_{i-1} & i \neq 1 \end{cases} \qquad A'_i = \begin{cases} 1 & i=1 \\ A'_{i-1}+B'_{i-1} & i \neq 1 \end{cases}$$

$$F_i = \left\{ \begin{array}{ll} 1 & i=1 \\ A'_{i-1}+B'_{i-1} & i \neq 1 \end{array} \right\} = A'_i = B'_{i+1}$$

or to every leaf (`Babies(1)` and `Adults(1)`):

$$B''_i = \begin{cases} 1 & i=1 \\ A''_{i-1} & i \neq 1 \end{cases} \qquad A''_i = \begin{cases} 1 & i=1 \\ A''_{i-1}+B''_{i-1} & i \neq 1 \end{cases}$$

$$F_i = \left\{ \begin{array}{ll} 1 & i=1,2 \\ A''_{i-2}+B''_{i-2} & i > 2 \end{array} \right\} =$$

$$= \left\{ \begin{array}{ll} 1 & i=1 \\ A''_{i-1} & i \neq 1 \end{array} \right\} = B''_i$$

In both of these variants only the base cases in the mutually recursive functions are modified. The function that implements F_n simply subtracts an offset to the argument with which it calls the mutually recursive functions, and modifies its base case to assure that such argument is positive.

In another interesting variant every recursion tree node is counted (`Babies(i)` and `Adults(i)`, $\forall i$). This time the recursive cases in the mutually recursive functions also need to be modified in order to return an additional value of 1:

$$B'''_i = \begin{cases} 1 & i=1 \\ 1+A'''_{i-1} & i \neq 1 \end{cases}$$

$$A'''_i = \begin{cases} 1 & i=1 \\ 1+A'''_{i-1}+B'''_{i-1} & i \neq 1 \end{cases}$$

In order to derive the final formula for F_n, first consider the following well known identity:

$$S_n = \sum_{i=1}^{n} F_i = F_{n+2} - 1$$

A simple change of variables gives:

$$F_n = S_{n-2} + 1 \tag{5}$$

Additionally, several identities can be deduced with a simple examination of the recursion tree:

$$S_n = B'''_n = 1 + A'''_{n-1} = 2 + A'''_{n-2} + B'''_{n-2} \tag{6}$$

Finally, combining (5) and (6) we can formulate descriptions

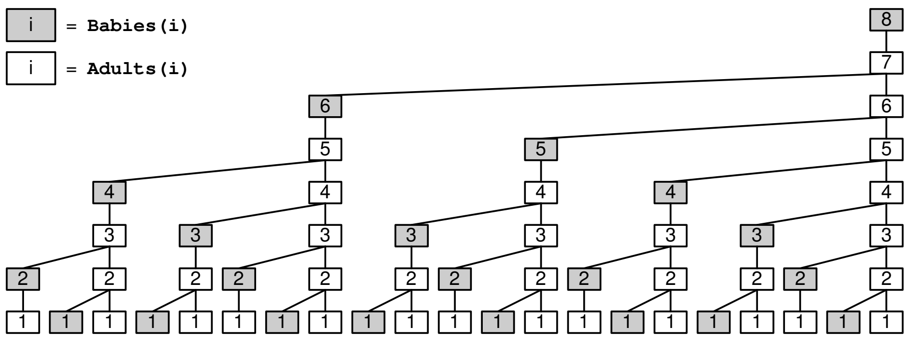


Figure 5: Recursion tree for `Fib_Mutual_Rec(6)` ≡ `Adults(7)` ≡ `Babies(8)`.

of the Fibonacci function:

$$F_i = \left\{ \begin{array}{ll} 1 & i=1,2 \\ 2 & i=3 \\ 3 & i=4 \\ 3+A'''_{i-4}+B'''_{i-4} & i>4 \end{array} \right\} =$$

$$= \left\{ \begin{array}{ll} 1 & i=1,2 \\ 2 & i=3 \\ 2+A'''_{i-3} & i>3 \end{array} \right\} = \left\{ \begin{array}{ll} 1 & i=1,2 \\ 1+B'''_{i-2} & i>2 \end{array} \right\}$$

Other similar variants can also be found. For example, the total number of calls to `Babies()`, or the total number of calls to `Adults()`+1 are Fibonacci numbers.

3.2 Computational complexity

The complexity in space (depth of the recursion tree) of the proposed algorithms is linear with respect to their parameter n. Their complexity in time (nodes of the recursion tree $\leq S_n$) is exponential with respect to n, in particular $\theta(\phi^n)$. Note that the number of nodes S_n is a linear function of F_n, which converges to $\frac{1}{\sqrt{5}}\phi^n$ [9]. Therefore, our algorithms and the classical recursive one share the same computational complexity. Note that the latter executes $2F_n - 1$ recursive calls to the `Fib_Rec` function.

4. DISCUSSION AND CONCLUSIONS

This paper proposes an attractive alternative for the calculation of Fibonacci numbers. Our approach is based on two mutually recursive functions, a strategy that we have not seen before in the literature. Instead of focusing on the mathematical recurrence relation associated with the Fibonacci sequence, the algorithm is derived from an analysis of Fibonacci's rabbits problem. This change of perspective has eventually led to a completely different algorithm, which might be welcomed with interest by students, especially those without a solid mathematical background. We believe that problems like the one presented in this paper help students understand the basic principles of recursion and induction, and can motivate them to search for recursive solutions to everyday problems. For example, the Fibonacci numbers are related to the golden ratio, since $\lim_{n\to\infty} \frac{F_{n+1}}{F_n} = \phi$. Due to the relationship of ϕ with many processes in nature, other interesting algorithms can emerge from different settings [7].

Finally, mutual recursive algorithms are generally difficult to understand, and therefore not treated in depth in many programming courses. The algorithm presented in this paper not only constitutes an appealing example, but also provides a different view of the Fibonacci numbers problem, and can therefore complement the study of multiple recursion. An empirical research aiming to evaluate the effectiveness of this algorithm in a classroom environment is currently one of our major pending projects.

5. ACKNOWLEDGEMENTS

This work is supported by project TIN2004-07568 of the Spanish Ministry of Education and Science.

6. REFERENCES

[1] M. D. Atkinson. The cyclic towers of Hanoi. *Information Processing Letters*, 13(3):118–119, 1981.

[2] R. S. Bird. Tabulation techniques for recursive programs. *ACM Computing Surveys*, 12(4):403–417, 1980.

[3] P. Buneman and L. S. Levy. The towers of Hanoi problem. *Information Processing Letters*, 10(4,5):243–244, 1980.

[4] B. Gosper and G. Salamin. HAKMEM. Technical Report AI Memo 239, Item 12, MIT, February 29 1972.

[5] R. Honsberger. *Mathematical Gems III*. Mathematical Association of America, 1985.

[6] D. E. Knuth. *The Art of Computer Programming*, volume 1-3 Boxed Set. Addison-Wesley, September 1998.

[7] D. R. Mack. The magical Fibonacci number. *IEEE Potentials*, 9(3):34–35, October 1990.

[8] O. Martín-Sánchez and C. Pareja-Flores. A gentle introduction to algorithm complexity for CS1 with nine variations on a theme by Fibonacci. *ACM SIGCSE Bulletin*, 27(2):49–56, 1995.

[9] I. Stojmenovic. Recursive algorithms in computer science courses: Fibonacci numbers and binomial coefficients. *IEEE Transactions on Education*, 43(3):273–276, August 2000.

[10] D. Wells. *The Penguin Dictionary of Curious and Interesting Numbers*. Penguin Books, Middlesex, England, 1986.