On the Number of Recursive Calls of Recursive Functions

Yannis Manolopoulos
Data Engineering Laboratory
Department of Informatics, Aristotle University
Thessaloniki, 54006 Greece
manolopo@delab.csd.auth.gr

ABSTRACT

The advantages and disadvantages of recursion are early introduced to students. Simplicity in coding but time and space inefficiency during execution are the main characteristics. In many occasions, recursive formulae lead to recursive functions/procedures that are highly inefficient as calls with the same parameters are executed several times. Here, we elaborate on a previous report [2], where a generalized analysis is carried out to derive the number of recursive calls of a recursive formula, the calculation of the Fibonacci numbers in particular. Here we re-examine the problem using a different and simpler approach, which generalizes as well.

1. INTRODUCTION

It is well known frmm introductory courses in Computer Programming, Data Structures and Algorithmics that recursive formulae may be easily coded and understood. On the other hand it is also true that recursion demands extra space overhead for stack maintenance as well as it implies extra time overhead in comparison to iteration since function calls are expensive operations compared to arithmetic or logic operations. Even worse is the fact that in many occasions direct application of a recursive formula into a recursive code results in a highly inefficient program because function calls with the same parameters are executed several times.

Motivation of the present report is the article of John Robertson in the June 1999 issue of Inroads [2], where an analysis is carried out to derive the number of recursive calls of a recursive formula using difference equations with initial conditions, or discrete dynamical systems (DDS). The author shown that there is a linear relationship between the Fibonacci numbers themselves and the number of recursive calls. He also demonstrated how this relationship generalizes to any type of DDS of second-order and DDS of higher-order.

Here we re-examine the problem of deriving the number of recursive calls of a recursive function by using a different and much simpler approach. Our approach generalizes equally well and can be easily introduced to students.

2. FIBONACCI NUMBERS

The driving example of [2] is the calculation of Fibonacci numbers. Given the recurrence relation:

$$F(n) \;=\; F(n-1) + F(n-2) \qquad n = 2, 3, ...$$

with initial conditions $F(0)$=0, and $F(1)$=1, the respective recursive algorithm can be easily coded as follows.

```
     function fib(n)
1.   if n<=1 then return n
2.   else return fib1(n-1)+fib1(n-2)
```

The author in [2] does not solve this recurrence but he connects this recurrence with the following recurrence:

$$G(n) \;=\; G(n-1) + G(n-2) + 1$$

which expresses the number of recursive calls. He assumes (without justification) that $G(n)$ is a linear function of $F(n)$ and, finally, he concludes that:

$$G(n) \;=\; 2F(n) - 1$$

Thus, the problem we are focusing here is the derivation of $G(n)$ for the case of Fibonacci numbers as well as for other similar (i.e. recursive in nature) problems. Our approach can better understood by remarking the following figure.

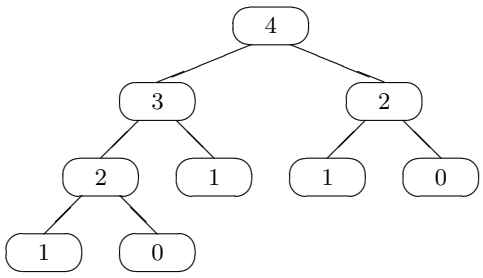

Figure 1: Calls to calculate F(4).

This figure illustrates a tree with nodes depicting the parameter values of the recursive calls. To understand the specific order of these recursive calls, someone has to traverse the tree using dfs. Now, the derivation of the total number of recursive calls comes from the remark that the probability distribution of the number of recursive calls of each specific Fibonacci number follows a certain pattern. The following

list gives the number of recursive calls for each specific Fibonacci number as it appears in the exaple of the figure.

$$\begin{gathered} F_4 \text{ is called } 1{=}F_1 \text{ time} \\ F_3 \text{ is called } 1{=}F_2 \text{ times} \\ F_2 \text{ is called } 2{=}F_3 \text{ times} \\ F_1 \text{ is called } 3{=}F_4 \text{ times} \\ F_0 \text{ is called } 2{=}F_3 \text{ times} \end{gathered}$$

Based on this observation, we are going to prove the following proposition.

Theorem.
The probability distribution of the number of recursive calls for each specific Fibonacci number in the case of calculating F_n (where $n > 1$) is as follows:

$$\begin{gathered} F_n \text{ is called } F_1 \text{ time} \\ F_{n-1} \text{ is called } F_2 \text{ times} \\ F_{n-2} \text{ is called } F_3 \text{ times} \\ \dots \\ F_1 \text{ is called } F_n \text{ times} \\ F_0 \text{ is called } F_{n-1} \text{ times} \end{gathered}$$

Proof.
We will prove the theorem by double induction. Obviously, the proposition holds for $n = 2$ as F_2 will be called $1 = F_1$ time, F_1 will be called $1 = F_2$ time, and F_0 will be called $1 = F_1$ time. Also, the proposition holds for $n = 3$ as F_3 will be called $1 = F_1$ time, F_2 will be called $1 = F_2$ time, F_1 will be called $2 = F_3$ times, and F_0 will be called $1 = F_2$ time. We suppose that for $n = k{-}2$ the following probability distribution of the number of recursive calls per each specific Fibonacci number will hold:

$$\begin{gathered} F_{k-2} \text{ is called } 1 = F_1 \text{ time} \\ F_{k-3} \text{ is called } 1 = F_2 \text{ times} \\ F_{k-4} \text{ is called } 2 = F_3 \text{ times} \\ \dots \\ F_1 \text{ is called } F_{k-2} \text{ times} \\ F_0 \text{ is called } F_{k-3} \text{ times.} \end{gathered}$$

Also, we assume that for for $n = k - 1$ the following probability distribution of the number of recursive calls per each specific Fibonacci number will hold:

$$\begin{gathered} F_{k-1} \text{ is called } 1 = F_1 \text{ time} \\ F_{k-2} \text{ is called } 1 = F_2 \text{ times} \\ F_{n-3} \text{ is called } 2 = F_3 \text{ times} \\ \dots \\ F_1 \text{ is called } F_{k-1} \text{ times} \\ F_0 \text{ is called } F_{k-2} \text{ times.} \end{gathered}$$

We have to prove that the probability distribution of the number of recursive calls per each specific Fibonacci number is as follows:

$$\begin{gathered} F_k \text{ is called } 1 = F_1 \text{ time} \\ F_{k-1} \text{ is called } 1 = F_2 \text{ times} \\ F_{k-2} \text{ is called } 2 = F_3 \text{ times} \\ \dots \\ F_1 \text{ is called } F_k \text{ times} \\ F_0 \text{ is called } F_{k-1} \text{ times.} \end{gathered}$$

This comes easily if we remark that we have to take into account the two previous probability distributions plus we have to add an extra function call for the F_k case. □

Now that we have derived the probability distribution function of the number of recursive calls when calculating the n-th Fibonacci number, we can add the elements of this probability distribution and get the desired number: i.e. the total number of recursive calls. The following corollary closes the case.

Corollary.
The total number of recursive calls during the calculation of the n-th Fibonacci number is $2F_{n+1} - 1$.

Proof.
Based on the previous theorem, we have to prove the relation:

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

After some simplifications based on Fibonacci properties, equivalently it comes up that we have to prove that:

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

This relation will be proved by induction. Apparently, it holds for $n = 1$. We assume that it holds for $n = k$, and we will prove that it holds for $n = k + 1$. In the other words, the following relation has to be true:

$$\sum_{i=1}^{k+1} F_i = F_{k+3} - 1$$

Departing form the left hand side we have:

$$\sum_{i=1}^{k+1} F_i = \sum_{i=1}^{k} F_i + F_{k+1} = F_{k+2} - 1 + F_{k+1} = F_{k+3} - 1$$

□

We can reach the previous result on the total number of recursive calls via a different path. We have proven that in order to calulate the n-th Fibonacci number, F_1 will be calculated F_n times, whereas F_0 will be calculated F_{n-1} times. Thus, the number of leaves of the respective tree will be equal to $F_{n-1} + F_n = F_{n+1}$. To derive the total number of recursive calls (i.e. the total number of nodes), we have to derive the total number of internal nodes. The following theorem is fundamental and can be found in standard Data Structures books. We include the proof for completeness.

Theorem.
For any binary tree with $n > 0$ nodes, if n_i denotes the number of nodes of degree i (for $0 \leq i \leq 2$), then:

$$n_0 \;=\; n_2 + 1$$

Proof.
It is apparent that:

$$n \;=\; n_0 + n_1 + n_2$$

Except the root, every node is connected by a branch, thus:

$$n \;=\; k + 1$$

where k is the number of branches. Also, every branch stems from a mode of degree 1 or 2. Thus:

$$k \;=\; n_1 + 2n_2$$

From these three expressions the theorem comes with simple algebra. In the present case $n_1 = 0$, however, deliberately we did not simplify the proof. □

Thus in our case, the number of recursive calls is $n_0+n_2 = 2n_0-1$, which means that the total number of recursive calls is equal to $2F_{n+1} - 1$. This way, we have reached the same result with [2], however, in a much simpler way from the mathematical and pedagogical point of view.

3. OTHER EXAMPLES

This derivation for the number of recursive calls during the Fibonacci number calculation can be easily used in other cases as well. For example, let us examine the following algorithm for the calculation of the n-choose-k combination.

```
    function Comb(n,k);
1.  if (k=0) or (k=n) then return 1
2.  else return Comb(n-1,k-1)+Comb(n-1,k)
```

The following tree depicts the recursive calls for the calculation of the 4-choose-2 combination. The node contents stand for the parameter values. Again, the number of leaves equals the value of the 4-choose-2 combination as the latter number is calculated by summing 1s from line 1 of the code fragment. Therefore, the total number of recursive calls equals twice the 4-choose-2 combination minus one.

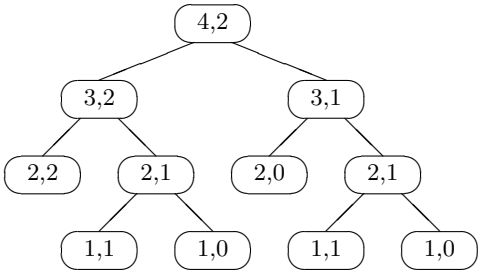

Figure 2: Calls to calculate the 4-choose-2 combination.

The following fragment is an abstraction of the mergesort algorithm [3]. Again, as can be easily derived form a similar figure, the number of recursive calls is twice the number of table elements minus one $(2n - 1)$.

```
    procedure merge_sort(left,right);
1.  if left<right then
2.     middle <-- (left+right) div 2;
3.     merge_sort(left,middle);
4.     merge_sort(middle+1,right);
5.     merge(left,middle,right)
```

As a final example, consider the following fragment which is an effort to calculate the power a^b (for integer $b > 0$) based on a divide-and-conquer philosophy. (For a discussion on the exponentiation problem see [1, 3].)

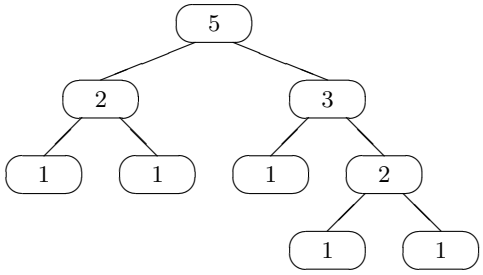

Figure 3: Calls to calculate a^5.

```
    function power(a,b)
1.  if b=1 then return a
2.  else return power(a,b/2)*power(a,(b+1)/2)
```

The same way, we remark here that the number of leaves equals the number that line 1 is executed (which is equal to the exponent b). Therefore, the total number of recursive calls equals twice the exponent minus one $(2b - 1)$.

4. GENERALIZATION

The recurrence relations of the examined examples of Fibonacci numbers, combination calculation, mergesort and exponentiation are second order in the sense that the right-hand side contains two smaller instances than the instance of the left-hand side. What happens in case of a recurrence relation of third order, e.g. of the form:

$$T(n) \;=\; T(n-1) + T(n-2) + T(n-3)$$

with $T(0) = T(1) = T(2) = 1$; In [2] the function $D(n)$, standing for the number of recursive calls, equals:

$$D(n) \;=\; D(n-1) + D(n-2) + D(n-3) + 1$$

and by assuming a linear function (without justification) of $D(n)$ from $T(n)$ it is concluded that:

$$D(n) \;=\; 3T(n)/2 - 1/2$$

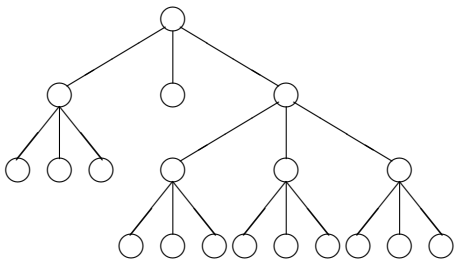

Figure 4: Ternary tree.

Here, we follow a different approach based on the previous theorem for binary trees.

Corollary.
For any ternary tree with $n > 0$ nodes, if n_i denotes the number of nodes of degree i (for $0 \le i \le 3$), then:

$$n_0 \;=\; 2n_3 + 1$$

Proof.
It is apparent that:

$$n \;=\; n_0 + n_3$$

Except the root, every node is connected by a branch, thus:

$$n \;=\; k + 1$$

where k is the number of branches. Also, in our case (see figure with the ternary tree), every branch stems from a mode of degree 3. Thus:

$$k \;=\; 3n_3$$

From these three expressions the corollary comes with simple algebra. □

Thus the total number of recursive calls is:

$$n_0 + n_3 \;=\; n_0 + \frac{n_0 - 1}{2} \;=\; \frac{3n_0}{2} - \frac{1}{2}$$

Thus, we have reached the same result with [2] via a simpler way.

5. CONCLUSIONS

Motivation to this report was the article by Robertson [2] on the number of recursive calls of a recursive formula. Here we re-examine the problem using a different and much simpler approach.

6. REFERENCES

[1] Brassard G. and Bratley P.: *Fundamentals of Algorithmics*, Prentice Hall, 1996.

[2] Robertson J.: How Many Recursive Calls a Recursive Function Make; *ACM SIGCSE Bulletin Inroads*, Vol.31, No.2, pp.60-61, 1999.

[3] Weiss M.A.: *Data Structures and Algorithm Analysis in C++*, Benjamin/Cummings, 1993.