Study of Recursive and Iterative Approach on Factorial and Fibonacci Algorithm

Vatsal Shah[1], Jayna Donga[2]

[1]*Assist. Prof., IT Dept. , BVM Engineering College, V.V.Nagar,vatsal.shah@bvmengineering.ac.in*

[2]*Assist. Prof., Computer Dept. ,MBICT, V.V.Nagar, jaynadonga@yahoo.com*

Abstract: Algorithm is wide area for research. In part of algorithm is solved various real time problem like job scheduling, shortest path and Eight Queen etc. For finding a solution multiple approaches work on single problem. In this paper we work on that direction. There are two main basic algorithm can be implemented in different nature. In this paper we study factorial and Fibonacci algorithm solved by simple iterative approach and recursive approach. At last we compare number of parameter like their number of operation, memory utilization and Time complexity.

Keyword: Iterative, Recursive, Time complexity

I. INTRODUCTION

Algorithm is collection of finite set of un ambiguous instruction that occur in sequence. lots of approaches available in algorithm for solving a problem. We discuss basically two approaches are iterative and recursive. Recursion is an important problem solving and programming technique and there is no doubt that it should be covered in the first year introductory computer science courses, in the second year data structure course, and in the third year design and analysis of algorithms course. While the advantages of using recursion are well taught and discussed in textbooks, we discovered that its potential pitfalls are often neglected and never fully discussed in literature. For the purpose of our discussion, we shall divide recursive functions into linear and branched ones[1]. Linear recursive functions make only one recursive call to itself. Note that a function's making only one recursive call to itself is not at all the same as having the recursive call made one place in the function, since this place might be inside a loop. It is also possible to have two places that issue a recursive call (such as both the *then* and *else* clauses of an *if* statement) where only one call can actually occur. The recursion tree of a linear recursive function has a very simple form of a chain, where each vertex has only one child. This child corresponds to the single recursive call that occurs. Such a simple tree is easy to comprehend, such as in the well known factorial function[3-4]. By reading the recursion tree from bottom to top, we immediately obtain the iterative program for the recursive one. Thus the transformation from linear recursion to iteration is easy, and will likely save both space and time. However, these savings are only in the constant of linear time complexity for both recursive and iterative solutions, and can be easily disregarded.

II. BACKGROUND THEORY

Iterative functions – are loop based imperative repetitions of a process (in contrast to recursion which has a more declarative approach). Iterative is part of Pseudo code and it is a language similar to a programming language used to represent algorithms. The main difference respect to actual programming languages is that pseudo code is not required to follow strict syntactic rules, since it is intended to be just read by humans, not actually executed by a machine[5].

Recursive function – is a function that is partially defined by itself and consists of some simple case with a known answer. A *recursive procedure* is a procedure that invokes itself. Also a set of procedures is called *recursive* if they invoke themselves in a circle, e.g., procedure *p*1 invokes procedure *p*2, procedure *p*2 invokes procedure *p*3 and procedure *p*3 invokes procedure *p*1. A *recursive algorithm* is an algorithm that contains recursive procedures or recursive sets of procedures[7]. Recursive algorithms have the advantage that often they are easy to design and are closer to natural mathematical definitions Example: Fibonacci number sequence, factorial function, quick sort and more. Some of the algorithms/functions can be represented in an iterative way and some may not.

The particular recursive algorithm for calculation Fibonacci series is less efficient. Consider the following situation of finding fib(4) through the recursive algorithm

```
int fib(n) :
     if( n==0 || n==1 )
        return n;
     else
        return fib(n-1) + fib(n-2)
```

Now when the above algorithm executes for n=4

```
          fib(4)

     fib(3)          fib(2)

  fib(2)   fib(1)     fib(1)   fib(0)

fib(1)  fib(0)
```

It's a tree. It says that for calculating fib(4) you need to calculate fib(3) and fib(2) and so on.

Notice that even for a small value of 4, fib(2) is calculated twice and fib(1) is calculated thrice. This number of additions grows for large numbers.

There is a conjecture that the number of additions required for calculating fib(n) is

```
fib(n+1) -1
```

So this duplication is the one which is the cause of reduced performance in this particular algorithm.

The iterative algorithm for Fibonacci series is considerably faster since it does not involve calculating the redundant things.

It may not be the same case for all the algorithms though.

III. IMPLEMENTATION AND RESULT ANALYSIS

In implementation point of view, We can use any of language for implementation. C programming is simple and we use for implementation of Fibonacci series and factorial using both approaches.

Factorial algorithm

Iterative approach of factorial

```
Iterative factorial (int num)
sum <-0
for  1 to num do
        sum<- sum+ i;
return sum;
End;
```

Recursive approach of factorial

```
Recursive factorial (int num)
if num=0
        return 1;
else
        return n*factorial(num-1);
end;
```

After implementation of factorial using both approach for taking input size starting from 10 to 100000.

N	Recursive	Iterative
10	334 ticks	11 ticks
100	846 ticks	23 ticks
1000	3368 ticks	110 ticks
10000	9990 ticks	975 ticks
100000	stack overflow	9767 ticks

Table 2: comparison of Iterative and recursive for factorial

The reason for the poor performance is heavy push-pop of the registers in the ill level of each recursive call.

Fibonacci Algorithm:

Iterative approach of Fibonacci

```
Integer fibon( Integer n)
    if ( $n \le 1$ )
        return n;
    else
        return fibon(n−1) + fibon(n−2);
```

Recursive approach of Fibonacci

```
Integer fibon( Integer n)
    if ( $n \le 1$ )
        return n;
    $b = 0$;
    $c = 1$;
    for ( $i = 2,3,\dots,n$ )    // $c = F_{i-1},\ b = F_{i-2},\ a = F_{i-3}$ (except when i=2).
        $a = b$;
        $b = c$;
        $c = b + a$;          // Now $c = F_i,\ b = F_{i-1},\ a = F_{i-2}$.
    return c;
```

After implementation of Fibonacci using both approach for taking input size starting from 5 to 100000.

N	Recursive	Recursive opt.	Iterative
5	5 ticks	22 ticks	9 ticks
10	36 ticks	49 ticks	10 ticks
20	2315 ticks	61 ticks	10 ticks
30	180254 ticks	65 ticks	10 ticks
100	too long/stack overflow	158 ticks	11 ticks
1000	too long/stack overflow	1470 ticks	27 ticks
10000	too long/stack overflow	13873 ticks	190 ticks
100000	too long/stack overflow	too long/stack overflow	3952 ticks

Table 2: comparison of Iterative and recursive for fibonacci

As before, the recursive approach is worse than iterative however, we could apply memorization pattern (saving previous results in dictionary for quick key based access), although this pattern isn't a match for the iterative approach (but definitely an improvement over the simple recursion).

IV. CONCLUSION

In this paper we discuss both approaches on different algorithm so we conclude that the matrix method of generating Fibonacci numbers is more efficient than the simple iterative algorithm, though in order to see its benefits, you will probably have to work with numbers consisting of hundreds of bits or more. We can directly use previous find value in to next steps. For memory purpose iterative is more appropriate compare to recursive. For smaller numbers, the simplicity of the iterative algorithm is preferable.

REFERENCE

[1] F. B. Chedid and T. Mogi, "A simple iterative algorithm for the towers of Hanoi problem," *IEEE Trans. Educ.*, vol. 39, pp. 274–275, May 1996.
[2] R. L. Kruse, C. L. Tondo, and B. P. Leung, *Data Structures and Program Design in C*. Englewood Cliffs, NJ: Prentice-Hall, 1997.
[3] A. B. Tucker, A. P. Bernat, W. J. Bradley, R. B. Cupper, and G. W. Scragg, *Fundamentals of Computing I*. New York: McGraw-Hill, 1994.
[4] T. L. Naps and D. W. Nance, *Introduction to Computer Science: Programming, Problem Solving, and Data Structures*. St. Paul, MN:West, 1995.
[5] E. B. K offman, *Pascal*. Reading, MA: Addison-Wesley, 1995.
[6] T. A. Standish, *Data Structures, Algorithms, and Software Principles*. Reading, MA: Addison-Wesley, 1994.
[7] E. B. Koffman and B. R. Maxim, *Software Design and Data Structures in Turbo Pascal*. Reading, MA: Addison-Wesley, 1994.