International Journal of Hybrid Information Technology
Vol.6, No.6 (2013), pp.127-134
http://dx.doi.org/10.14257/ijhit.2013.6.6.11

Discussion on Writing of Recursive Algorithm

Song Jinping

Computer Department, Jining Teachers College, Wulanchabu, China

jnsongjinping@126.com

Abstract

Writing a program with the recursive method is a simple and effective way of program design, making the logic of the program concise and clear. This paper will focus on the analytical program, writing method of the recursive algorithm, as well as the optimization of recursive program. Formula method or mathematical induction can be applied for writing of the recursive algorithm more simply. Meanwhile, either of them could provide the method of tail recursive to deal with the problem lies in recursive algorithm that massive time and space of the system are occupied.

Keywords: *formula method; recursive algorithm; mathematical induction; tail recursive*

Recursion is widely used as an algorithm in programming language. Recursion refers to the repetitive phenomenon that the function, process or subroutine calls itself in the running course. As an important concept in computer science, recursion is a simple and understandable method often used in program design, and it can make the program concise and clear and reduce the amount of code.

1. Features of Recursive Programming

(1)Recursive exits. Recursive exit has defined the terminal conditions of the recursion. When the terminal conditions met during program execution, the recursion will terminate. Some issues may include several recursive exits.

(2)When the conditions are not met, the issue will be divided into several sub-issues according to the nature of the issue. The solution of the sub-issues will be achieved by revising the parameters to make the function call itself in certain ways. Then, the original issue can be solved by combining the solutions of the sub-issues. When the recursion calls itself, modification on parameters must be assured to meet the conditions of the recursive exits.

2. Key Points of Recursive Programming

2.1. Solve the problem from the overall level

The difficulty of recursion lies on that the human brain is not suitable to trail the process of calling itself in recursion, because the human brain has no stack used to memorize as a computer does. However, human beings can induce while computer knows only how to call and return. Therefore, we should comprehend and design recursion from the overall level. Mathematical induction and recursion are in a symmetrically related: the former constantly

ISSN: 1738-9968 IJHIT

expands itself while the latter disintegrates. Recursion is executed by 2 steps: divide the main problems into sub-problems, and get the solution step by step by solving the sub-problems.

The key point is to make the best of assumption of the mathematical induction, i.e., assuming the solutions to the sub-issues have been presented. Once you do that, you are trying to solve the problem from the overall level without being troubled by the details any longer. When designing a recursive program, we should firstly design an algorithm framework as a common program, control the program logic and handle the recursive call as if we are calling another designed function, about which we need only to know what it does and then return. About the details, we can just ignore. It seems to be sort of paradoxical because in fact, the function is calling itself, so why we do not need to know how it works? That's the reason why recursion is beyond comprehension. Therefore, we have to put the details aside until we have confirm the framework.

2.2. The returned value of a function

After having designed the framework in the first step, we should pay attention to the details including process branch and returned value of a function. Returned value of a function directly decides if the function works properly, because the recursive sub-programs will call what the function returns, and the returned value of the recursive sub-programs will affect the final result. Thus, we must concern with the returned value of a function. When the returned result of the sub-program is used by a caller, the caller will return. That's where the problem exists: the consistency of the returned value of the function. Generally, the design of the recursive function which is a little complex involves many logic branches whose returned values (the solution of the function in different situation) must be in accordance with each other. What is error prone is that the recursive theory is based on the sub-issue and the sub-issue is a small-scale parent issue. Since we assume the sub-issue is solvable and the solution to the parent issue comes from the combination of the solution to the sub-issue. In this way, we know that both the parent issue and the sub-issue aim to deal with the same problem and their result should be identical.

3. Writing Method of Recursive Program

3.1 Using the formula method to write a recursive program

Programming includes two stages: logic design and implementation stage. Logic design relies on the mathematical thinking to define the algorithm regardless of the programming language and the implementation environment. The algorithm can be demonstrated using natural language or flow chart. If the recursive formula of the algorithm can be obtained in the logic stage, at least the following advantages can be produced:

(1)Clear Separation of logic stage with implementation stage can largely simplify the program design.

(2)The mathematical method can make the derivation of the recursive formula much easier than with other methods.

(3)Since formula is the most accurate and simplest way to describe an algorithm, with the help of the recursive formula, coding will be very simple and the readability of the program will be better.

The formula method of the recursive program design should firstly express the issue with the recursive function in the mathematical meaning. Then, the writing of the recursive program is the direct translation of the formula.

As in case 1, the mathematical formula of the n!-level to the natural number n is as follows:

$$\begin{cases} n! = 1 & (n = 1) \\ n! = n * (n-1)! & (n > 1) \end{cases} \quad (1)$$

Thus, when n=1, n! = 1, when n > 1, n! = n*(n-1)!

The recursive program is:

```
1  fac (int n)
2  {
3  if (n == 1)   return 1;
4  return n* fac (n - 1) ;  /*  recursive part  */
5  }
```

Case 2 is for Fibonacci series, and the mathematical formula is as follows:

A positive integer n is given,

$$\begin{cases} F(n) = 0 & (n = 0) \\ F(n) = 1 & (n = 1) \\ F(n) = F(n-1) + F(n-2) & (n > 1) \end{cases} \quad (2)$$

That is, the first number is 0, and the second is 1, each latter number is the sum of the two previous numbers.

The recursive program is:

```
1  long fib (int n)
2  {
3  if ( n = = 0)  return 0;
4  else if( n = = 1 )
5  return 1;
6  else
7  return fib (n-1) + fib (n-2);
8  }
```

From the above examples, we can know that formula is a simple and effective designing theory which centralizes the difficulties of the programming and the program comprehension to the recursive mathematical formula. What else, we know that this programming thought can simplify the programming and the given programs are more understandable than the common program. This thought possesses universality which can be used in most recursive program. The program designed with the recursive formula owns standard branch structure which makes the writing and the comprehension much easier.

3.2 Writing a recursive program with mathematical induction

Mathematical induction is an important proof method in math. When proving a mathematical law, the mathematical induction thought follows the following procedure: firstly, prove the law by substituting simple numbers; then, in the assumption that a certain number N conforms with the law, prove that N+1 is also acceptable. In fact, the mathematical induction uses recursive principle, which can be vividly called the Domino Theory. Because if N+1 is acceptable, it can be proved that all the numbers are acceptable by recursion forward or backward.

In addition, recursion also uses the recursive law. In the whole program, the same law will be used repetitively. In this point, recursion and mathematical induction are essentially identical. Therefore, usually the mathematical induction can be used to design the recursive program. People usually say, "Computer is a branch of mathematical application", and this is best reflected in this case.

Similar to the above case which aims to get the n!-level of the natural number n:

It is given that n!=n*(n-1)*(n-2)*(n-3)*…*2*1

Firstly, we know that when n=1, n!=1

Secondly, it is given that R(n)=n!, R(n+1)=(n+1)!

Thirdly, solve the relation between R(n+1) and R(n). R(n)=n!,sinceR(n+1)=(n+1)!= (n+1)*(n)*(n-1)*…*2*1=(n+1)*n!=(n+1)*R(n)即：R(n+1)=(n+1)*R(n) => R(n)=n*R(n-1)

Now, a function is drafted according to this formula:

```
1  fac (int n)
2  {
3  return n * fac (n - 1) ;  /* recursive part*/
4  }
```

Then, the ending part is supplemented. This part will be used only once in the whole process, without which the process will recur endlessly.

The function is changed as follows:

```
1  fac (int n)
2  {
3  if (n = = 1) return 1;
4  return n * fac (n - 1);  /* recursive part */
5  }
```


From the aforementioned examples, it is learned that we are familiar with the mathematical induction when analyzing the problem. When writing a recursive program, we should reach the recursive ending part according to the first step of the mathematical induction. Then, we can create the recursive part of the function in line with the third step. The recursive solving process is to find out the relation between R(n) and R(n+1).

Now we are going to use the mathematical induction to write a recursive problem: the classical Hanoi Tower.

Hanoi Tower: there are three uprights (named A, B, C; A is the upright where the disk first located, B is the target upright, and C is used as the auxiliary upright), n disks of different diameters are moved one after another from A to B. It is required only one disk can be moved each time and moved among the three uprights. In addition, the bigger disk cannot be put on the small ones.

Firstly, analysis is conducted using mathematical induction:

(1)When there is only one disk, we can make sure the unique motion: move the disk directly from A to B.

(2)Assuming that there are 3 disks on A, we can finally move these disks to B (or to C) as required, recursive process as shown in Figure 1.

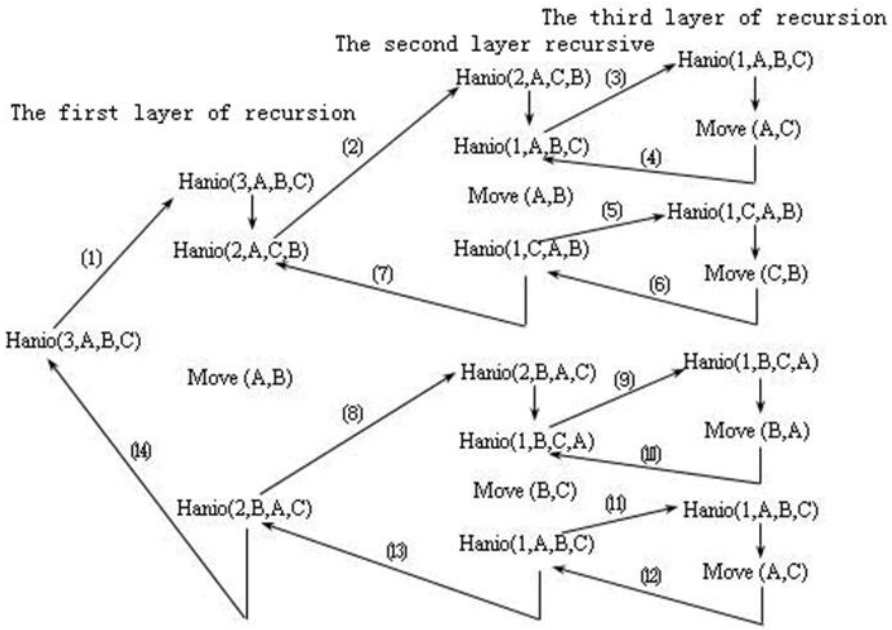

Figure 1. Hanoi Algorithm is a recursive process

(3)Then, we can prove that when there are n+1 disks, we can also move them all to B as required: because we can firstly move the n disks to C (the second step has assured), next to move the last one left to B, then move the disks on C to B.

Design the program according to the recursive function design steps we have summarized:

Firstly, confirm the ending part: when there is only one disk moving, we can move it to B directly. That is: if (n = = 1) mov (n, A, B) (the mov here indicates to move the disk numbered n from A to B).

Secondly, confirm the recursive part (the relation between n+1 and n). That is, move n disks to C (assured in step two), move the last one left to B, and then move the disks on C to B. Now, we are going to transfer the words to program:

Assuming Hanoi (int n, int A, int C, int B) is the Hanoi function required. It means move the n disks piled together from small to big according to their diameter from A to B as required, and C is the aid upright.

The code is as follows:

Hanoi (n-1, A, C, B); /*firstly, move n-1 disks to C*/

mov (n, A, B); /*move the last disk left behind to B*/

Hanoi (n-1, C, B, A); /*move the disks on C to B*/

The second step is finished, and finally the function is composited：

```
1  void  Hanoi (int n, int A, int B, int C)
2  {
3  if (n = = 1)
4  mov (n, A , C);
5  else
6  {
7  Hanoi (n-1, A, C , B);  /*recursive part*/
8  mov (n, A, B);
9  Hanoi (n-1, B,A,C); /*recursive part*/
10  }
11  }
```

The greatest advantage of using the mathematical induction to design the recursive program is that it can help the designer to get rid of over consideration on recursion, because the code you have designed must have implied the recursive steps. You can get the code by transforming the words directly.

4. Optimization of the Recursive Program

4.1 Analysis of the working of the recursive or non-recursive program

Recursive program is not only an effective programming method, but also a valid method to analyze problems. However, the recursive process and the information saved in the recursive process differ from that in the non-recursive program. In the non-recursive program, when calling the non-recursive sub-issues, the system has to save two kinds of information:

(1)The return address after calling the sub-programs

(2)The local variable value used to call the sub-programs

After carrying out the called sub-program and before returning to the main program, the system should first recover the local variable value of the called sub-program, and then get back to the return address of the called sub-program.

When calling the recursive function, what the system does is the same in form with what the systems does when calling the non-recursive program, but the contents and methods of saving the information are totally different. The saved information of each recursive call makes up a job stack, usually including:

(1)The local variable value of the recursive call;

(2)The return address, *i.e.*, the address of the following ones after the call statements in the recursive process;

(3)The actual parameters which combine with the formal parameters in the calling, including function name, reference parameters, numerical values, and *etc.*

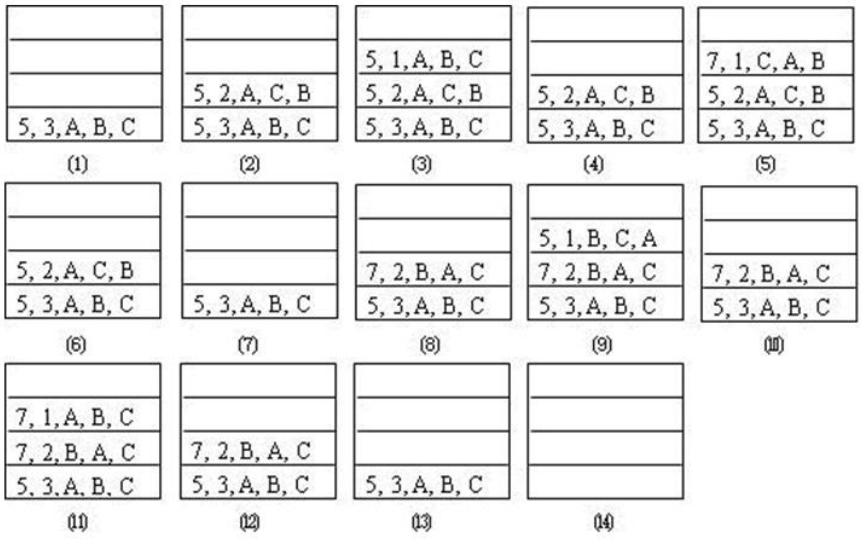

Figure 2. Hanoi Algorithm execution work stack profile

When returning to the recursion, the system firstly uses the information saved later. Therefore, the system often sets up stacks to save the recursive call information. The recursive program has the two following defects when working: on the one hand, the recursive program calls too many layers that waste relatively more operating time: on the other hand, the recursive program occupies too much storage space, and it may even create the system paralysis. Therefore, most programmers are unwilling to use the recursive method.

4.2 Optimization of the recursive program

We call this recursive method mentioned above linear recursion, and there exists another recursive method, i.e. tail recursive method, which can effectively improve the defects of the linear recursion. Take the above Fibonacci sequence as an example; here we will not start from the definition of the Fibonacci sequence but from the regular process of producing a sequence. The sequence produces 0 and 1, which is easy enough and returns directly, and the following counting process is to accumulate. In the process of recursion, we should hold the mode, and in this mode three numbers must be kept, i.e. the front two numbers a, b and the iterated step number c. Thus, our method is defined as follows:

```
1  int fib(int n,int a,int b,int c)
2  {  if (n<3)
```

```
3       return 1;
4   else
5       if (n= =c)
6          return a+b;
7       else
8           return fib(n,a=b,b=a+b,c=c+1);/* recursive part */
9   }
```

Since this method has kept the calculated status of the last time in each recursion, we refer it as linear iterative process, also known as tail recursion. That's to say, the last thing the function does is to call the function, this is called ending call.

Since each calculation has kept the status, eliminating the redundant calculation. The efficiency of this method is apparently higher than the former one. No matter how deep the recursion goes, the size of the stack will remain the same. It can be found that the process and the cycling of the tail recursion is basically equal. Since we can conveniently substitute the process of the tail recursion with the cycling, many languages have provided the optimization in the write-level to tail recursion, *i.e.*, to transfer tail recursion to cycling codes in the writing stage. However, it also makes sense to the language which provides no optimization of the tail recursion, for example, in the Fibonacci function which uses tail recursion, the one that can be called by Fib(1001) runs fast, and it does not appear stack overflow until it is called by Fib(1002) . However, if we use the linear method to calculate the program when n=30, the speed shall obviously decrease. When it is above 40, it is going to die.

From the above discussion, it is learned that not all the recursive programs operate poorly. If we adopt tail recursion, the efficiency of the recursive program can also be high.

References

[1] W. Yan and W. Wu, Eds., "Data Structure", Tsinghua University Press, Beijing, **(2002)**.
[2] H. Tan, Ed., "C Programming", Tsinghua University Press, Beijing, **(2002)**.
[3] X. Dan, *et al.*, Ed., "Application Research on Implementation of Non - Recursive Method for Recursive Problem", Computer and Modernization, **(2011)**.
[4] Z. Zhu and C. Zhu, "Recursive Algorithm for Non-Recursive Implementation", Microcomputer System, vol. 3, **(2003)**.

Authors

Song Jinping

She received double bachelor's degrees from Inner Mongolia Normal University, China, in 1998, she is an assistant professor in Computer Department, Jining Teachers College, China. Her research interests include embedded system and computer science education.

