Recursion VS Iteration (Looping) : Speed & Memory Comparison

Recursion VS Iteration (Looping) : Speed & Memory Comparison

Recursive functions – is a function that partially defined by itself and consists of some simple case with a known answer. Example: Fibonacci number sequence, factorial function, quick sort and more.
Some of the algorithms/functions can be represented in iterative way and some may not.

Iterative functions – are loop based imperative repetition of a process (in contrast to recursion which has more declarative approach).

Comparison between Iterative and Recursive approaches from performance considerations

Factorial:

```
//recursive function calculates n!
static int FactorialRecursive(int n)
{



 if (n <= 1) return 1;



 return n * FactorialRecursive(n - 1);



}
//iterative function calculates n!
static int FactorialIterative(int n)
{



 int sum = 1;
```

```
    if (n <= 1)

        return sum;

    while (n > 1)

    {

        sum *= n; n--;

    }

    return sum;

   }
```

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

As we can clearly see the recursive is a lot slower than the iterative (considerably) and limiting (stackoverflow).

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

Fibonacci:

```
//--------------- iterative version ---------------------
static int FibonacciIterative(int n)
{

 if (n == 0)

     return 0;

 if (n == 1)

     return 1;

 int prevPrev = 0;

 int prev = 1;

 int result = 0;
 for (int i = 2; i <= n; i++)

 {

    result = prev + prevPrev;

    prevPrev = prev;

    prev = result;

 }

 return result;

}
//--------------- naive recursive version ---------------------
static int FibonacciRecursive(int n)
{

 if (n == 0)

    return 0;
 if (n == 1)

    return 1;

 return FibonacciRecursive(n - 1) + FibonacciRecursive(n - 2);

}
//--------------- optimized recursive version ---------------------
static Dictionary resultHistory = new Dictionary();
static int FibonacciRecursiveOpt(int n)
{

 if (n == 0)

    return 0;

 if (n == 1)

    return 1;

 if (resultHistory.ContainsKey(n))

    return resultHistory[n];
 int result = FibonacciRecursiveOpt(n - 1) + FibonacciRecursiveOpt(n - 2);



 resultHistory[n] = result;



 return result;



}
```

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

As before the recursive approach is worse than iterative however, we could applymemoization [http://en.wikipedia.org/wiki/Memoization] pattern (saving previous results in dictionary for quick key based access), although this pattern isn't match for iterative approach (but definitely improvement over the simple recursion).

Now let's think about when it is a good idea to use recursion and why. In many cases there will be a choice: many methods can be written either with or without using recursion.

Q: Is the recursive version usually faster?
A: No — it's usually slower (due to the overhead of maintaining the stack)

Q: Does the recursive version usually use less memory?
A: No — it usually uses **more** memory (for the stack).

Q: Then **why** use recursion??
A: It makes the code beautiful – recursion is a beauty of programming.
Sometimes it is much simpler to write the recursive version.

Posted 19th January 2011 by Eslam Ghanem