Iteration via Tail Recursion

Friday, November 3, 2006

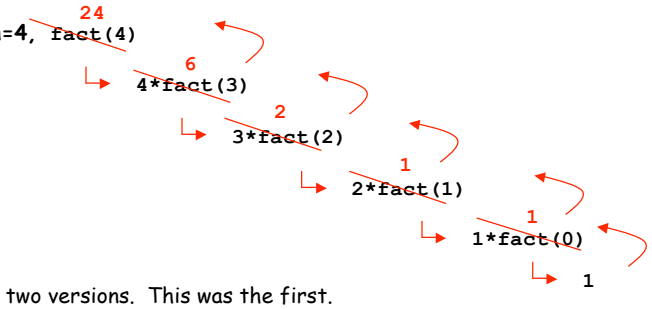
CS111 Computer Programming

Department of Computer Science
Wellesley College

Recall factorial n! = n x (n-1)!

```
public static int fact (int n)
{
     if (n == 0) {
          return 1;
     } else {
          return n * fact (n - 1);
     }
}
```

So, if n=4, fact(4)



*We wrote two versions. This was the first.

Calculating Factorial Iteratively

step	num	ans
1	4	1
2	3	4
3	2	12
4	1	24
5	0	24

- An iteration is characterized by a collection of state variables (here num and ans) that change over time.
- The execution of an iteration can be summarized by an iteration table, where each row represents the state at one point in time, and the next row can be determined from only the previous row.
- An iteration typically maintains a loop invariant: something that is true for each row in the table. Here the invariant is:

  num * ans = fact(num0)

- The answer of an iteration can be computed from the last row.

Iteration via Tail Recursion

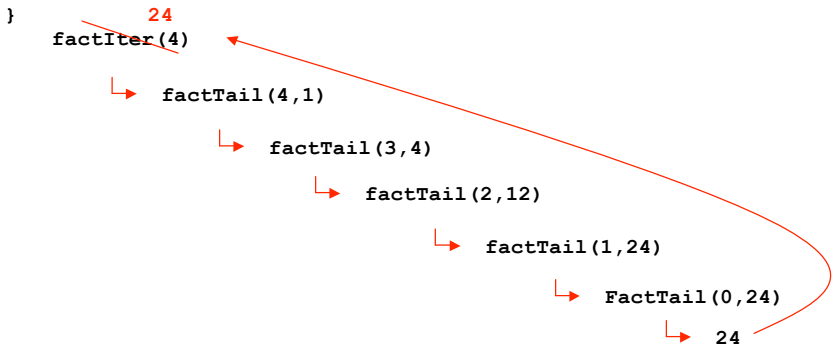
```
public static int factTail (int num, int ans) {
    if (num == 0) {
        return ans;
    } else {
        return factTail(num-1, num*ans); // No pending operations!
    }
}
```

- Tail recursion is recursion without the "glue" step. There are no pending operations.
- To solve a problem iteratively, it is often necessary to have a "wrapper" method that invokes the tail-recursive method on the initial values. E.g.:

```
public static int factIter (int num, int ans) {
    return factTail(n,1);
}
```

`factIter(n)` invokes `factTail(n,1)`

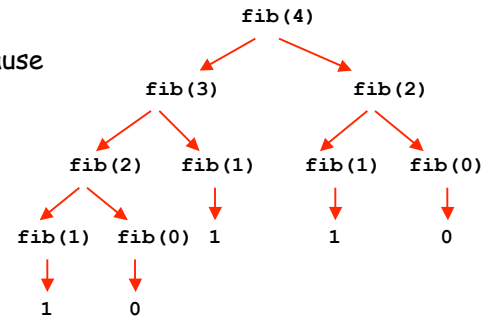
```
public static int factTail (int num, int ans)
{
     if (num == 0) {
          return ans;
     } else {
          return factTail(num-1, num*ans);
     }
}
```



Remember Fibonacci?

```
public int fib(int n) {
   if (n < 2) // Assume n >= 0
      return n;
   else
      return fib(n-1) + fib(n-2);
}
```

This is inefficient because it computes the same subtrees many times.



Iteration leads to a more efficient Fib

The Fibonacci sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, …

Table for calculating the 8th Fibonacci number:

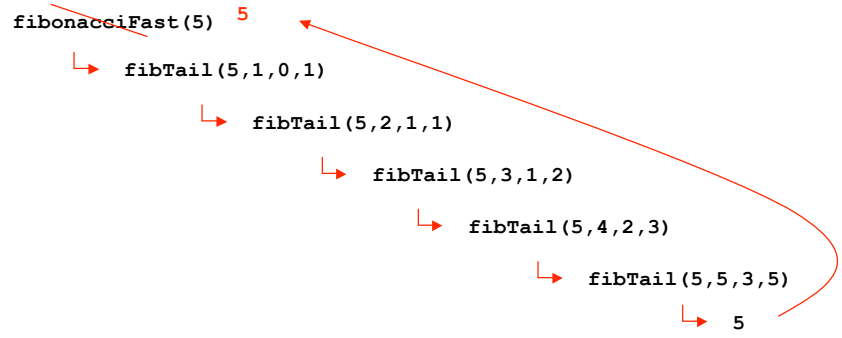
step	n	i	fibIMinus1	fibI
1	8	1	0	1
2	8	2	1	1
3	8	3	1	2
4	8	4	2	3
5	8	5	3	5
6	8	6	5	8
7	8	7	8	13
8	8	8	13	21

Iterative Fibonacci in Java

```
public static int fibIter(int n) {
   if (n==0) {

   } else {

   }
}

public static int fibTail(int n, int i, int fibIMinus1, int fibI) {
   if (          ) {

   } else {

   }
}
```

fibonacciIter(n) invokes fibTail(n,1,0,1);

```
public int fibTail(int n, int i, int fibIMinus1, int fibI)
{
     if (i == n) {
          return fibI;
     } else {
          return fibTail(n, i+1, fibI, fibIMinus1 + fibI);
     }
}
```



Iterative List Reversal

Table for reversing [1,2,3,4,5]

step	oldPile	newPile
1	[1,2,3,4,5]	[]
2	[2,3,4,5]	[1]
3	[3,4,5]	[2,1]
4	[4,5]	[3,2,1]
5	[5]	[4,3,2,1]
6	[]	[5,4,3,2,1]

Iterative List Reversal in Java

```
public static IntList reverseIter (IntList list) {

}

public static intList reverseTail(IntList oldPile, IntList newPile) {









}
```