Ben Sottile

3/3/2021

CS-331-01

Algorithms Assignment – Recursion

* Create two functions to find Fibonacci numbers
  + int FibonacciSeries (int n) – a non-recursion based implementation
  + int FibonacciRecursive(int n) – a recursion based implementation
    - For each n is the desired Fibonacci number: e.g. n = 1 -> 0, n = 2 -> 1, n = 3 -> 1, n = 4 -> 2, n = 5 -> 3, n = 6 -> 5, etc.
  + For the recursive version identify the four parts described above

My recursive algorithm int FibonacciRecursive(int n) is as follows:

public static int FibonacciRecursive(int n){  
 if(n == 1) {  
 return 0; // 1. The direct solution in which the recursion terminates  
 }else if (n==2){  
 return 1; // 1. The direct solution in which the recursion terminates  
 }else {  
 return *FibonacciRecursive*(n - 1) + (*FibonacciRecursive*(n - 2));   
 // 2. The problem division in which the original problem is reduced to one or more “smaller” problems (approaching the case for the direct solution)  
// 3. The recurrence, the self referential call  
// 4. The recombination where the solutions to the “smaller” problems are combined into the final solution This happens on the stack  
 }  
 }  
}

The algorithm is relatively straight forward and most the action happens in that final call. When the algorithm is called it immediately goes into the if statement. If I wanted to find the 6th fibonnacci number it would run like this:

Fibonacci 6th

N=6

If(n==1)? No

Else if (n ==2) No

Return Fibonacci Recursive(n-1)(5) + Fibonacci Recursive(n-2)(4) ( This is the recurrence, the self referential call)( This is also part of the problem division in which the original problem is reduced to one or more “smaller” problems (approaching the case for the direct solution) (The sum of fib 6 is equal to the sum of FibonacciRecursive(5) and FibonacciRecursive(4) When values FibonacciRecursive(4)=2 and FibonacciRecursive(5)=3 evaluate this is pulled from the stack to evaluate and finally return FibonacciRecursive(6) as 2 + 3 = 5. This is part of the recombination where the solutions to the “smaller” problems are combined into the final solution

Go to stack

Fibonacci 5th

N=5

If(n==1)? No

Else if (n ==2) No

Return Fibonacci Recursive(n-1)(4) + Fibonacci Recursive(n-2)(3) ( This is the recurrence, the self referential call) (The sum of fib 5 is equal to the sum of FibonacciRecursive(4) and FibonacciRecursive(3))( This is also part of the problem division in which the original problem is reduced to one or more “smaller” problems (approaching the case for the direct solution)When values FibonacciRecursive(4)=2 and FibonacciRecursive(3)=1 evaluate this is pulled from the stack to evaluate FibonacciRecursive(5) as 2 + 1 = 3. This is part of the recombination where the solutions to the “smaller” problems are combined into the final solution

Go to stack

Fibonacci 4th

N=4

If(n==1)? No

Else if (n ==2) No

Return Fibonacci Recursive(n-1)(3) + Fibonacci Recursive(n-2)(2) ( This is the recurrence, the self referential call) (The sum of fib 5 is equal to the sum of FibonacciRecursive(3) and FibonacciRecursive(2) )( This is also part of the problem division in which the original problem is reduced to one or more “smaller” problems (approaching the case for the direct solution)When values FibonacciRecursive(2)=1 and FibonacciRecursive(3)=1 evaluate this is pulled from the stack to evaluate FibonacciRecursive(4) as 1 + 1 = 2. This is part of the recombination where the solutions to the “smaller” problems are combined into the final solution

Go to stack

Fibonacci 3th

N=3

If(n==1)? No

Else if (n ==2) NoReturn Fibonacci Recursive(n-1)(2) + Fibonacci Recursive(n-2)(1)( This is the recurrence, the self referential call) (The sum of fib 3 is equal to the sum of FibonacciRecursive(2) and FibonacciRecursive(1)) )( This is also part of the problem division in which the original problem is reduced to one or more “smaller” problems (approaching the case for the direct solution) When values FibonacciRecursive(2)=1 and FibonacciRecursive(1)=0 evaluate this is pulled from the stack to evaluate FibonacciRecursive(3) as 1 + 0 = 1. This is part of the recombination where the solutions to the “smaller” problems are combined into the final solution

Go to stack

Fibonacci 2th

N=2

If(n==1)? No

Else if (n ==2) No

Return 1;

Fibonacci 1th

N=1

If(n==1)? Yes

Return 0;

This is the direct solution in which the recursion terminates FibonacciRecursive(2)=1 and FibonacciRecursive(1)=0 evaluate this allows for the other functions which rely on these recursions to be found and completed.

Describe the general layout of a recursive algorithm to implement the 8-queens problem

* + You do not need to write the program – that is optional
  + Identify the four parts as described above

A general layout of a recursive solution to the Queens Problem might look like this.

nQueens(target,Q+1, Boolean[target][target] holding queens,String[target][target] holding queens)

if(target == Queens)

// 1. General Solution where recursion terminates

Return String[target][target] holding queens

else{

if you can add a queen

//The problem division is adding one queen at a time and taking away a queen if you can’t add them.

//The recombination happens as the problem goes on. The final recombination happens at the end.

add safe queen to safe solution(randomly pick a spot from the boolean

Alter Boolean and string arrays and

//The Recursion - self Referential call

return nQueens(target,Q+1, Boolean[target][target] available loacations,String[][])).

Else if you can’t add a safe queen nQueen remove a x from the string array and turn true any Booleans made available

//The Recursion - self Referential call

return nQueens(target,Q-1, Boolean[target][target] available loacations) ,String[target][target])

}