Taller Recursión

Estudiante: Juan Sebastian Getial Getial
Código: 202124644

Asignatura: Fundamentos de Programación Funcional y Concurrente
Docente: Juan Francisco Diaz Frias

Universidad del Valle
Santiago de Cali
2022

Informe de Procesos

Función mcdTFA

Esta función genera un **proceso recursivo lineal**, ya que solo realiza un llamado recursivo por cada ejecución y al finalizar todos los llamados recursivos realiza una operación de multiplicación, por tanto, la función no es lo último en realizarse.

Función mcdEBez

Esta función ejecuta un **proceso iterativo**, puesto que no realiza ninguna otra operación luego de finalizar los llamados recursivos, por lo que la función es la última en ejecutarse.

Función fibonacciA

Al ejecutar esta función se produce un **proceso recursivo de árbol**, debido a que en el cuerpo de la función se realizan más de un llamado recursivo, en este caso 2, y al finalizar la evaluación de cada llamado recursivo se efectúa una operación de suma, por consiguiente la función no es lo último en realizarse.

Función fibonacciI

Al llamar a esta función se realiza un **proceso iterativo**, dado que luego de realizar los llamados recursivos no se realiza ninguna operación extra, por lo que el último llamado recursivo de la función es lo último en realizarse.

Informe de Corrección

Argumentación sobre la corrección

1.1 Maximo comun divisor

1.1.1 Teorema Fundamental de la Aritmética

Por definición(**Def**) se tiene que:

$$\forall n \in N \exists k \in N \exists p^1, p^2, \ldots, p^k \in N | primo(p^1) \wedge \ldots \wedge primo(p^k) :$$
$$(p^1 \leq p^2 \leq \ldots \leq p^k) \wedge (n = p^1 p^2 \ldots p^k)$$

En base a lo anterior, dados:

$$n = p_1^{i1} p_2^{i2} \ldots p_k^{ik} \text{ y } m = p_1^{j1} p_2^{j2} \ldots p_k^{jk}$$

se tiene que:

$$\mathbf{mcd(n, m) = p_1^{min(i1,j1)} p_2^{min(i2,j2)} \ldots p_k^{min(ik,jk)}}$$

mcdTFA

```
def mcdTFA(ln:List[Int], lm:List[Int], primos:List[Int]) : Int = {
  if (primos.tail.isEmpty) Math.pow(primos.head, Math.min(ln.head, lm.head)).toInt
  else Math.pow(primos.head, Math.min(ln.head, lm.head)).toInt * mcdTFA(ln.tail, lm.tail, primos.tail)
}
```

mcdTFA(ln:List[Int], lm:List[Int], primos:List[Int]) ==

$$\mathbf{mcdTFA((n_1,...,n_k), (m_1,...,m_k), (p_1,...,p_k))}$$

donde primos, como su nombre lo indica, es una lista de números primos ordenados de forma ascendente y se tiene dos números(m y n) tal que:

$$n = p_1^{n1} \ldots p_k^{nk} \text{ y } m = p_1^{m1} \ldots p_k^{mk}$$

Teo :

A continuación se demostrará que:

$\forall$ **ln,lm,primos** $\in$ **List[Int] :**

mcdTFA(ln:List[Int], lm:List[Int], primos:List[Int]) == mcd(n, m)

lo cual, gracias al TFA, se puede ver a su vez como:

$$\mathbf{mcdTFA((n_1,...,n_k), (m_1,...,m_k), (p_1,...,p_k)) == p_1^{min(n1,m1)} \ldots p_k^{min(nk,mk)}}$$

Caso Base: mcdTFA((i), (j), (p)) == $p^{min(i,j)}$

Demostración Caso Base:

mcdTFA((i), (j), (p))

→ if(() == ()) $p^{min(i,j)}$ else $p^{min(i,j)}$ * mcdTFA((), (), ())

→ **$p^{min(i,j)}$** ♦

Caso Inducción:

Se tomará como hipótesis que el programa funciona para listas de k elementos y se demostrara que funciona para lista de k+1 elementos.

Hipótesis: mcdTFA((n_1,...,n_k), (m_1,...,m_k), (p_1,...,p_k)) == $p_1^{min(n1,m1)}$...$p_k^{min(nk,mk)}$

Hipótesis→mcdTFA((n_0,...,n_k),(m_0,...,m_k),(p_0,...,p_k)) ==

$p_0^{min(n0,m0)}$...$p_k^{min(nk,mk)}$

Demostración Caso Inducción:

mcdTFA((n_0,...,n_k), (m_0,...,m_k), (p_0,...,p_k))

→ if((p_1,...,p_k) == ()) $p_0^{min(n0,m0)}$

else $p_0^{min(n0,m0)}$ * mcdTFA((n_1,...,n_k), (m_1,...,m_k), (p_1,...,p_k))

→ $p_0^{min(n0,m0)}$ * **mcdTFA((n_1,...,n_k), (m_1,...,m_k), (p_1,...,p_k))**

Hip→ $p_0^{min(n0,m0)}$ * **$p_1^{min(n1,m1)}$...$p_k^{min(nk,mk)}$**

→ **$p_0^{min(n0,m0)}$...$p_k^{min(nk,mk)}$** ♦

Por lo que se tiene finalmente que:

mcdTFA((n_1,...,n_k), (m_1,...,m_k), (p_1,...,p_k)) == $p_1^{min(n1,m1)}$...$p_k^{min(nk,mk)}$

qué, como se mencionó en un principio, se puede ver como:

∀ln,lm,primos ∈ List[Int] :

mcdTFA(ln:List[Int], lm:List[Int], primos:List[Int]) == mcd(n, m)

1.1.2 Algoritmo de la División

Por definición(**Def**) se tiene que:

mcd(n, m) = {

m = 0 → n

Si, por algoritmo de la división **n = mq + r → mcd(m, r)==mcd(m,n%m)**

}

mcdEBez

```scala
def mcdEBez(n:Int, m:Int) : (Int,Int,Int) = {
  def mcdBZ(r1:Int, r2: Int, x1:Int, x2:Int, y1:Int, y2:Int) : (Int, Int, Int) = {
    if(r2 == 0) (r1,x1,y1)
    else {
      val q = Math.round(r1/r2)
      mcdBZ(r2, r1%r2, x2, x1 - (q*x2), y2, y1 - (q*y2))
    }
  }
  mcdBZ(n, m, 1, 0, 0, 1)
}
```

Teo:

A continuación se demostrará que:

$\forall$ **n,m** $\in$ **N : mcdEBez(n, m) == (mcd(n,m), x, y) , mcd(n,m) = nx + my**

Estado(S_i): (r1, r2, x1, x2, y1, y2)

Estado Inicial(S_0): (n, m, 1, 0, 0, 1)

Estado Final(S_f): (r1, 0, x1, x2, y1, y2) → r2 == 0

Respuesta(S_f): (r1, x1, y1)

Inv((r1, r2, x1, x2, y1, y2)):

n*x2 + m*y2 == r2 **(1)**

^

n*x1 + m*y1 == r1 **(2)**

^

mcd(n, m) == mcd(r1, r2) **(3)**

Trans((r1, r2, x1, x2, y1, y2)):(r2, r1%r2, x2, x1 - [Int(r1/r2)*x2], y2, y1 - [Int(r1/r2)*y2])

1. Inv(S_0)

Demostración:

Inv(S_0)

→ Inv((n, m, 1, 0, 0, 1))

→ n*0 + m*1 == m ^ n*1 + m*0 == n ^ mcd(n, m) == mcd(n, m)

→ **m == m ^ n == n ^ mcd(n, m) == mcd(m, m)** ♦

2. $S_i \neq S_f$ ^ Inv(S_i) → Inv(Trans(S_i))

Hipótesis: Inv((r1, r2, x1, x2, y1, y2)) ^ r2 ≠ 0

Hipótesis → Inv(Trans(S_i))

Demostración:

Inv(Trans(S_i))

→ Inv(Trans((r1, r2, x1, x2, y1, y2)))

→ Inv((r2, r1%r2, x2, x1 - [Int(r1/r2)*x2], y2, y1 - [Int(r1/r2)*y2]))

(1)→ n*(x1 - [Int(r1/r2)*x2]) + m*(y1 - [Int(r1/r2)*y2]) == r1%r2

→ n*x1 - n*x2*Int(r1/r2) + m*y1 - m*y2*Int(r1/r2) == r1%r2

→ -Int(r1/r2)***(n*x2 + m*y2)** + **(n*x1 + m*y1)** == r1%r2

Hip→ -Int(r1/r2)***r2** + **r1** == r1%r2

Algoritmo de la División

→ **r1 == r1%r2 + Int(r1/r2)*r2** ♦

(2)→ n*x2 + m*y2 == r2

Hip→ **n*x2 + m*y2 == r2** ♦

(3)→ mcd(n, m) == mcd(r2, r1%r2)

Hip→ mcd(n, m) == mcd(r2, r1%r2) **^ r2 ≠ 0**

Def→ mcd(n, m) == mcd(r2, r1%r2) **== mcd(r1, r2)**

Hip→ **mcd(n, m) == mcd(r1, r2)** ♦

→ **Inv(Trans(Si))** ♦

3. Inv(S_f) → Respuesta(S_f) == (mcd(n, m), x, y) , nx + my = mcd(n, m)

Demostración:

Inv((r1, 0, x1, x2, y1, y2))

→ n*x2 + m*y2 == 0 ^ **n*x1 + m*y1 == r1 ^ mcd(n, m) == mcd(r1, 0)**

→ n*x1 + m*y1 == r1 ^ mcd(n, m) == mcd(r1, 0)

Def→ n*x1 + m*y1 == r1 ^ mcd(n, m) == mcd(r1, 0) **== r1**

→ **n*x1 + m*y1 == r1 ^ mcd(n, m) == r1**

→ Respuesta((r1, 0, x1, x2, y1, y2))

→ (r1, x1, y1)

→ **(mcd(n, m), x1, y1) ^ n*x1 + m*y1 == mcd(n, m)** ♦

4. Por otro lado, en cada paso la componente m del estado es más pequeña. Por tanto, está cada vez más cerca de 0. En consecuencia, después de n iteraciones llega a 0. Esto implica que:

mcdEBez(n, m) == mcdBZ(n, m, 1, 0, 0, 1) == (r1, x1, y1)

== (mcd(n, m), x, y) , mcd(n, m) = nx + my

Finalmente se tiene que:

∀n,m ∈ N : mcdEBez(n, m) == (mcd(n,m), x, y) , mcd(n,m) = nx + my

1.2 Números Fibonacci

Por **definición(Def)** de fibonacci se tiene que:

fib(0) = 1

fib(1) = 1

fib(n) = fib(n-1) + fib(n-2)

1.2.1 fibonacciA

```
def fibonacciA(n:Int) : Int = {
  if (n == 0 || n == 1) 1
  else fibonacciA(n-1) + fibonacciA(n-2)
}
```

Teo:

A continuación se demostrará que:

∀n ∈ N : fibonacciA(n) == fib(n)

Caso Base 1: fibonacciA(0) == fib(0) == 1

Demostración Caso Base 1:

fibonacciA(0)

→ if (0 == 0 || 0 == 1) 1 else fibonacciA(0-1) + fibonacciA(0-2)

→ **1** ♦

Caso Base 2: fibonacciA(1) == fib(1) == 1

Demostración Caso Base 2:

fibonacciA(1)

→ if (1 == 0 || 1 == 1) 1 else fibonacciA(1-1) + fibonacciA(1-2)

→ **1** ♦

Caso Inducción:

Hipótesis: fibonacciA(n) == fib(n)

Hipótesis → fibonacciA(n+1) == fib(n+1)

Demostración Caso Inducción:

fibonacciA(n+1)

→ if (n+1 == 0 || n+1 == 1) 1 else fibonacciA(n+1-1) + fibonacciA(n+1-2)

→ **fibonacciA(n) + fibonacciA(n-1)**

Hip→ fib(n) + fib(n-1)

Def→ **fib(n+1)** ♦

Por lo que se puede concluir que:

∀n ∈ N : fibonacciA(n) == fib(n)

1.2.2 fibonaccil

```
def fibonacciI(n: Int): Int = {
  def fibI(m:Int, x:Int, y:Int) : Int = if (m > n) y else fibI(m+1, x+y, x)
  fibI(1,1,1)
}
```

Teo:

A continuación se demostrará que:

∀n ∈ N : fibonaccil(n) == fib(n)

Estado(S_i): (m, x, y)

Estado Inicial(S_0): (1, 1, 1)

Estado Final(S_f): (n+1, x, y) → m == n+1

Respuesta(S_f): y

Inv((m, x, y)): x == fib(m) ^ y == fib(m-1)

Trans((m, x, y)): (m+1, x+y, x)

1. Inv(S_0)

Demostración:

Inv(S_0)

→ Inv((1, 1, 1))

→ 1 == fib(1) ^ 1 == fib(0)

Def→ True ♦

2. ($S_i \neq S_f$ ^ Inv(S_i)) → Inv(Trans(S_i))

Hipótesis: Inv((m, x, y)) == (x == fib(m)) ^ (y == fib(m-1)) ^ m ≠ n+1

Hipótesis → Inv(Trans(S_i))

Demostración:

Inv(Trans(S_i))

→ Inv(Trans((m, x, y)))

→ Inv((m+1, x+y, x))

→ **x+y** == fib(m+1) ^ **x == fib(m)**

Hip→ fib(m) + **fib(m-1)** == fib(m+1) ^ **true**

→ fib(m) + fib(m-1) == fib(m+1)

Def→ true ♦

3. Inv(S_f) → Respuesta(S_f) == fib(n)

→ Inv((n+1, x, y))

→ x == fib(n+1) ^ y == fib(n+1-1)

→ **y == fib(n)**

→ Respuesta(S_f)

→ **y**

→ **fib(n)** ♦

4. Finalmente, en cada paso la componente m del estado es uno más. Por tanto, está cada vez más cerca de n + 1. En consecuencia, después de z iteraciones llega a n + 1. Esto implica que:

fibonaccil(n) == fibl(1, 1, 1) == y == fib(n)

Finalmente se tiene que: ∀**n** ∈ **N : fibonaccil(n) == fib(n)**

Casos de Prueba

Función mcdTFA

1. mcdTFA(List(3), List(0), List(2)) = **1** = mcd(8,1)
2. mcdTFA(List(1,2,1,1), List(2,1,2,1), List(2,3,5,7)) = **210** = mcd(630,2100)
3. mcdTFA(List(1,1,0,1,0), List(1,0,2,0,1), List(2,3,5,7,11)) = **2** = mcd(42,550)
4. mcdTFA(List(2,0,0,0,1), List(2,1,0,0,0), List(2,3,5,7,11)) = **4** = mcd(44,12)
5. mcdTFA(List(1,0,1,0,0,0,0,0,1), List(2,0,1,1,0,0,0,0,0),
   List(2,3,5,7,11,13,17,19,23)) = **10** = mcd(230,140)

Los casos de prueba anteriores evidencian un buen funcionamiento de la función con diferentes valores que necesitan un número determinado de primos. En algunos casos se muestra la complejidad a la hora de pasar como argumento números que necesitan un primo muy lejano al 2, debido a que la lista de primos aumenta y por consiguiente, las demás listas también lo hacen, por lo que la velocidad de ejecución se ve afectada. Aunque, ese detalle no afecta al resultado final, solo a la velocidad de ejecución, por lo que se puede decir que los casos bases refuerzan la correccion de la funcion.

Función mcdEBez = (mcd(n,m), x, y) , mcd(n,m) = nx + mx

1. mcdEBez(5,0) = **(5, 1, 0)**
2. mcdEBez(44,12) = **(4,-1,4)**
3. mcdEBez(230, 140) = **(10,-3,5)**
4. mcdEBez(963,657) = **(9,-15,22)**
5. mcdEBez(30263,20657) = **(1,7305,-10702)**

Estos casos de prueba reflejan la efectividad de la función con valores muy pequeños y grandes, los cuales se obtienen muy rápido debido a la naturaleza iterativa de la función. Por ello, se considera que son suficientes para reforzar la corrección de la función.

Función fibonacciA

1. fibonacciA(0) = **1**
2. fibonacciA(1) = **1**
3. fibonacciA(10) = **89**
4. fibonacciA(30) = **1346269**

5. fibonacciA(45) = **1836311903**

En los casos de prueba anteriores están incluidos los casos base de la función y otros casos que evidencian su buen funcionamiento. Aunque, cabe destacar que el tiempo de ejecución del último caso fue considerablemente alto, lo cual se debe al proceso recursivo de árbol que genera la función. Sin embargo, la respuesta fue correcta, por lo que se concluye que los casos son aptos para reforzar la corrección de la función.

Función fibonaccil

1. fibonaccil(0) = **1**
2. fibonaccil(1) = **1**
3. fibonaccil(10) = **89**
4. fibonaccil(30) = **1346269**
5. fibonaccil(45) = **1836311903**

Estos casos de prueba son los mismos casos de prueba de fibonacciA, los cuales en este caso también reflejan el buen funcionamiento de la función y refuerzan la corrección de la función, sin embargo, cabe destacar que la velocidad de ejecución fue mucho mejor en esta función que en la anterior, lo cual se hizo muy notorio en el último caso.

Por último, cabe señalar que tanto la función **fibonacciA** como la función **fibonaccil** no devuelven el valor correcto si se les da como parámetro un valor mayor a 45. Esto es debido a que son funciones que devuelven un valor tipo Int, el cual se encuentra entre -2147483648 y 2147483647, por lo que al recibir cualquier cualquier valor mayor a 45, la respuesta no es correcta, ya que el resultado excede el rango dado y por tanto, no es un valor tipo Int.

Conclusiones

Al comparar las funciones **mcdTFA** y **mcdEBez**, se pudo notar que **mcdTFA** presentaba una mayor complejidad a la hora de especificar los parámetros, debido a que, dependiendo del valor, las listas podian ser de un tamaño considerable, lo cual, debido al proceso recursivo lineal que genera, afectaría el rendimiento de la función. Por ello, **mcdEBez**, debido a su simpleza y velocidad de ejecución, gracias a su naturaleza iterativa, se considera más óptima.

Por otro lado, ocurre algo similar entre las funciones **fibonacciA** y **fibonaccil**, ya que **fibonacciA**, al generar un proceso recursivo de árbol, presenta una menor velocidad de ejecución en comparación con **fibonaccil**, que genera un proceso iterativo, lo cual se hace mucho más notorio cuando el parámetro es mayor o igual a 40. Por consiguiente, la función **fibonaccil** es mucho mejor para este caso.

Finalmente, luego de ver el funcionamiento de cada una de las funciones, se puede concluir que se obtuvo un mejor rendimiento por parte de las funciones que generan un proceso iterativo en comparación con las que generan un proceso recursivo lineal o de árbol, lo cual se pudo observar a simple vista en su velocidad de ejecución.