Rosalind: Algorithm Heights - Manual en Castellano

Alejandro Cebrián del Valle

19/08/2020

Sucesión Fibonacci

La sucesión Fibonacci (F_n) es una sucesión infinita que empieza con los numeros 0 y 1 que cumple la relación de recurrencia « *cada término es la suma de los dos anteriores* ». Fue descrita ya en el siglo XIII por Leonardo de Pisa, alias *Fibonacci*. La importancia de esta sucesión ha sido tal que ha fascinado desde hace mucho a personas tanto matemáticas como ha naturalistas[1]. La sucesión Fibonacci se obtiene:

$$F_n = \begin{cases} 0 & \text{Si } n = 0 \\ 1 & \text{Si } n = 1 \\ F_{n-2} + F_{n-1} & \text{Si } n > 1 \end{cases} \qquad \forall n \in N$$

Propiedades de la sucesión Fibonacci

Algunas propiedades de la sucesión Fibonacci son:

- El cociente entre un término F_n y F_{n-1}, esto es, su inmediatamente anterior, tiende a acercarse al número aureo ϕ (1.618033...), siendo este valor en el infinito.
- Cualquier número natural se puede escribir mediante la suma de un número limitado de términos de la sucesión de Fibonacci.
- Uno de cada tres números es múltiplo de 3; uno de cada cinco, es multiplo de 5; y en general, uno de cada m términos de la serie es divisible por m.
- Cada número de Fibonacci es el promedio del término que se encuentra dos posiciones antes y el término que se encuentra una posición después:

$$F_n = \frac{F_{n-2} + F_{n+1}}{2}$$

- El máximo común divisor de dos números de Fibonacci sualesquiera es otro número de Fibonacci
- La suma infinita de los término de la sucesión $\frac{F_n}{10}$ es $\frac{10}{89}$, siendo el 89 un número de Fibonacci, concretamente, el *décimo*.

Resolución

La sucesión Fibonacci puede ser abordada desde muchos campos de la computación. Se pueden obtener términos de la sucesión mediante recurrencia, mediante programación dinámica... En cualquier caso, en las siguientes secciones se verán varios encajes para resolver tanto la obtención de términos como saber el término dada una posición. De esta manera, se proponen tres soluciones:

[1] Para quien le guste, recomiendo este video de Derivando

- **Mediante función recursiva**: esto es, una función que dado un ordinal, da el número que corresponde a la serie (esto es, introducido el 8, da el octavo número de la serie). El funcionamiento de esta función es el que sigue: se evaluan tres resultados, si n es 0, devuelve 0; si n es 1, devuelve 1; y si es mayor que uno, la función devuelve la suma de la función del número anterior más el número antepenúltimo.

Input : Numero natural n
Result: n-ésima posición de secuencia Fibonacci

Fn *Fibonacci(n : int)*:
```
if n == 0 then
    return 0;
if n == 1 then
    return 1;
else
    return Fibonacci(n-1) + Fibonacci(n-2);
end
```
Algorithm 1: Pseudocódigo de la función recurrente.

- **Mediante programación dinámica**: Mediante una función, se crea un vector que almacena la secuencia Fibonacci hasta el término que se busca. Este modo de funcionamiento consume más memoria, pero ante la repetición, es más eficiente al no tener que volver a calcular el término cada vez que se ejecuta.

Input : Numero natural n
Result: n-ésima posición de secuencia Fibonacci

Fn *Fibonacci(n : int)*:
```
fArray ← [0, 1];
i ← 2;
for i → n + 1 do
    fArray[i] ← (fArray[i-1] + fArray[i-2]);
end
return fArray[n]
```
Algorithm 2: Pseudocódigo de la función de programación dinámica.

- **Mediante matrices**: existe una forma de calcular mediante la elevación a la n-ésima potencia de la matriz

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}$$

Input : Numero natural n
Result: n-ésima posición de secuencia Fibonacci

Fn *Fibonacci(n : int)*:
```
F ← ( 1 1 )
    ( 1 0 );
if n == 0 then
    return 0;
else
    M ← ( 1 1 )
        ( 1 0 );
    for i = 0 → n - 1 do
        F ← F * M = ( F_{0,0} * M_{0,0} + F_{0,1} * M_{1,0}   F_{0,0} * M_{0,1} + F_{0,1} * M_{1,1} )
                    ( F_{1,0} * M_{0,0} + F_{1,1} * M_{1,0}   F_{1,0} * M_{0,1} + F_{1,1} * M_{1,1} );
    end
    return F[0][0];
end
```
Algorithm 3: Pseudocódigo de la función que implementa calculo con matrices.

Resolución en Python

```
## Recursive Function to Fibonacci sequence
def recursiveFibonacciSeq(n):
   if n == 0:
     return 0
   elif n == 1:
     return 1
   else:
     return recursiveFibonacciSeq(n - 1) + recursiveFibonacciSeq(n - 2)

## Dynamic Programation
def dynamicPFibonacciSeq(n):
   fArray = [0,1]
   for i in range(2, n + 1, 1):
      fArray += [fArray[i - 1] + fArray[i - 2]]
   return fArray[n]

## Matrix calculus
def matrixFibonacciSeq(n):
   F = [[1, 1], [1,0]]
   if n == 0:
      return 0
   else:
      M = [[1, 1], [1,0]]
      for i in range(2, n):
         x = (F[0][0] * M[0][0] + F[0][1] * M[1][0])
         y = (F[0][0] * M[0][1] + F[0][1] * M[1][1])
         z = (F[1][0] * M[0][0] + F[1][1] * M[1][0])
         w = (F[1][0] * M[0][1] + F[1][1] * M[1][1])

         F[0][0] = x
         F[0][1] = y
         F[1][0] = z
         F[1][1] = w
   return F[0][0]
## Probe Section
print("Recusive:\t", recursiveFibonacciSeq(9), recursiveFibonacciSeq(22))
```

```
## Recusive:     34 17711
```

```
print("Dynamic programation:\t", dynamicPFibonacciSeq(9), dynamicPFibonacciSeq(22))
```

```
## Dynamic programation:     34 17711
```

```
print("Matrix:\t", matrixFibonacciSeq(9), matrixFibonacciSeq(22))
```

```
## Matrix:   34 17711
```

Resolución en R

```
## Recursive Function to Fibonacci sequence
recursiveFibonacciSequence <- function(n){
  if(n == 0) {
    return(0)
  }
  if(n == 1) {
    return(1)
  }
  else {
    return(recursiveFibonacciSequence(n-1) + recursiveFibonacciSequence(n-2))
  }
}
## Dynamic Programation
dynamicPFibonacciSequence <- function(n){
  fArray <- c(0, 1)
  for (i in 2:(n + 1)){
    fArray <- append(fArray, (fArray[i - 2] + fArray[i - 1]));
  }
  return(fArray[n + 1])
}
## Matrix calculus
matrixFibonacciSeq <- function(n){
  fMatrix <- matrix(c(1,1,1,0), nrow = 2, ncol = 2);
  if (n == 0){
    return(0)
  } else {
    mMatrix <- matrix(c(1,1,1,0), nrow = 2, ncol = 2);
    for (i in 2:(n - 1)){
      fMatrix <- fMatrix %*% mMatrix;
    }
    return(fMatrix[1][1])
  }
}
## Probe Section
cat(sprintf("Recusive:\t%d\t%d\n", recursiveFibonacciSequence(9),
            recursiveFibonacciSequence(22)))
```

```
## Recusive:    34  17711
```

```
cat(sprintf("Dynamic Programation:\t%d\t%d\n", dynamicPFibonacciSequence(9),
            dynamicPFibonacciSequence(22)))
```

```
## Dynamic Programation:    34  17711
```

```
cat(sprintf("Matrix:\t%d\t%d\n", matrixFibonacciSeq(9),
            matrixFibonacciSeq(22)))
```

```
## Matrix:  34  17711
```