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 cualesquiera 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**: 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.

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.

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[2] mediante la elevación a la n-ésima potencia de la matriz $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$, tal que:

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

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.

[2]Según lo visto en Proof of this result related to Fibonacci numbers? (2011, September 5); Mathematics Stack Exchange..

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
```

Búsqueda Binaria

La búsqueda binaria es un algoritmo del tipo « Dividir-y-Conquistar »[3]. Esta estrategia consiste en dividir el problema en problemas más pequeños a la misma instancia; resolver esos subproblemas y la combinación apropiada de las respuestas (Dasgupta, Papadimitriou, Vazirani; *Algorithms*; McGraw-Hill; 2006.). De esta manera, el problema se hace poco a poco en tres partes: particionando el problema en subproblemas, y así recursivamente hasta que el problema es totalmente resoluble; y la posterior aglutinación.

Para encontrar una clave o índice k en un largo archivo que contiene A[1,. . ., n] claves ordenadas, primeramente se compara k con la mediana del vector ordenado y, dependiendo de si es mayor o menor, se busca en la primera o segunda mitad del subvector que deja por encima o por debajo la mediana.

Existen distintas formas de aplicación de este tipo búsquedas, algunas de las cuales implican el uso de árboles binarios. En este caso, habrá que ceñirse a lo propuesto: dado un archivo de 4 lineas, la primera da el número de términos que que tiene el vector ordenado, la segunda el de orden aleatorio, la tercera el vector ordenado de números (rango de número desde $[-10^5, 10^5]$); y la cuarta y última el vector desordenado.

Se busca saber la posición en el vector ordenado de cada término de vector desordenado.

Resolución

Pseudocódigo

Result: Indice del término buscado

```
Fn binarySearch(array : array, item : int[,str,...]):
    lowestValue ← 0;
    maxValue ← array.lenth - 1;
    index = -1;
    while lowestValue <= maxValue do
        midValue = lowestValue + maxValue / 2;
        if array[midValue] == item then
            return midValue
        if item < array[midValue] then
            maxValue ← mid + 1;
        else
            lowestValue ← mid + 1;
        end
    end
```

Algorithm 4: Pseudocódigo de la implementación de la búsqueda binaria.

[3]Los fans del mundo latino, que creemos en las *limes* a este lado del Rhin y del Canal de la mancha, diriamos: *divide et vinces, divide ut imperes y divide ut regnes*

Resolución en Python

```
### Auxiliar function
def readRosalindArchive(nameFile):
   lines = []
   with open(nameFile, "r") as fileContent:
      for line in fileContent:
         lines.append(line.replace("\n", "").split())
   lines[0] = int("".join(lines[0]))
   lines[1] = int("".join(lines[1]))
   lines[2] = [int(x) for x in lines[2]]
   lines[3] = [int(y) for y in lines[3]]
   return lines
### Binary search function
def binarySearch(array, item):
   init = 0
   end = len(array) - 1
   index = -1
   while init <= end:
      mid = (init + end) // 2
      if item == array[mid]:
         index = mid
         break
      else:
         if item < array[mid]:
            end = mid -1
         else:
            init = mid + 1
   return index
### Probe Section
dataset = readRosalindArchive("rosalind-BinarySearch.txt")
results = []
for elem in dataset[3]:
  index = binarySearch(dataset[2], elem)
  if index != -1:
     index += 1
  results += [index]
print(" ".join([str(result) for result in results]))
```

```
## 4339 4147 2688 7864 -1 3707 4858 3933 5937 4969 1456 6602 169 5358 6962 3596 943 976 5218 7701 7698 1
```

Resolución en R

```
### Binary search function
binarySearch <- function(arraySearch, item){
  lowValue <- 0;
  maxValue <- length(arraySearch) - 1;
  index <- -1;
  while (lowValue <= maxValue){
    mid = (lowValue + maxValue) %/% 2;
    if (item == arraySearch[mid]){
      index = mid;
      break;
    } else {
      if (item < arraySearch[mid]){
        maxValue <- mid - 1;
      } else {
        lowValue <- mid + 1;
      }
    }
  }
  return(index);
}
### Probe Section
##### First, we read the archive. The Unique important section is the 3rd and
##### 4th line.
fileLines <- readLines("rosalind-BinarySearch.txt");
arrayOrdered <- as.numeric(unlist(strsplit(fileLines[3]," ")));
arrayUnordered <- as.numeric(unlist(strsplit(fileLines[4]," ")));
##### Second, lets search...
results <- c();
for (element in arrayUnordered) {
  results <- append(results, binarySearch(arrayOrdered, element));
}
cat(results)
```

```
## 4339 4147 2688 7864 -1 3707 4858 3933 5937 4969 1456 6602 169 5357 6962 3596 943 976 5218 7701 7698 1
```

ajfkajfkajf