Potenciación recursiva y Fibonacci O(log(N))

Potenciación recursiva

Entrada: Un valor numérico *X* y una potencia entera positiva *N*

Salida: X^N

```
r = X
for i = 2 to N:
    r *= X
print(r)
```

¿Cuál es la eficiencia de este algoritmo?

→ O(N) ¿Se puede hacer mejor?

Solución mediante Divide & Vencerás

```
function recursivePower(X, N):
    if N = 1:
        return X
    else if N%2 = 0:
        A = recursivePower(X, N/2)
        return A*A
    else:
        A = recursivePower(X, (N-1)/2)
        return X*A*A
```

¿Cuál es la eficiencia de este algoritmo?

→ O(log(N))

Fibonacci

Entrada: Un valor entero positivo *N*

Salida: *N*-ésimo término de la serie de Fibonacci

```
f[1] = 0, f[2] = 1
for i = 3 to N:
      f[i] = f[i-1] + f[i-2]
return f[N]
```

¿Cuál es la eficiencia de este algoritmo?

→ O(N) ¿Se puede hacer mejor?

Solución mediante Divide & Vencerás

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

Notemos que aunque se trata de multiplicación de matrices, siempre son 2x2, lo cual implica una cantidad constante de operaciones.

Prueba: Si $n = 1$, $\begin{bmatrix} F_2 & F_1 \\ F_1 & F_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^1$

Si $n > 1$, $\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} F_n & F_{n-1} \\ F_{n-1} & F_{n-2} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$

$$= \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^1$$

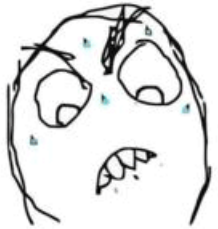
Solución mediante Divide & Vencerás

Juntando el teorema de la forma matricial de la serie con la potenciación recursiva, obtenemos:

```
function fib(N):
   (a', b', c', d') = fibMRP(a, b, c, d, N)
   return b'
```

¿Cuál es la eficiencia? O(log(N))

¿Se puede hacer mejor?



```
function fibMRP(a, b, c, d, N):
   if N = 1:
      return (1, 1, 1, 0)
   else if N%2 = 0
      (a', b', c', d') = fibMRP(a, b, c, d, N/2)
      return (a'*a'+b'c', b'(a'+d'), c'(a'+d'), d'*d'+b'c')
   else
      (a', b', c', d') = fibMRP(a, b, c, d, (n-1)/2)
      return (a'*a'+b'c' + b'(a'+d'), a'*a'+b'c', c'(a'+d') + d'*d'+b'c', c'(a'+d'))
```

Considerando $\begin{bmatrix} a & b \\ c & d \end{bmatrix}^k = \begin{bmatrix} a' & b' \\ c' & d' \end{bmatrix}$

Solución en un computador sin punto flotante

$$fibonacci(n) = \frac{\varphi^n - (1-\varphi)^n}{\sqrt{5}}$$

siendo $\varphi = \frac{1+\sqrt{5}}{2} = 1{,}618\ldots$ (conocido como número áureo)*

¿Cuál es la eficiencia? *O(log(N))* por tratarse de una potenciación, pero en este caso escalar (constantes menores)

Dado que no podemos contar con la precisión de esta solución, tendremos que conformarnos con la solución mediante potenciación recursiva en forma matricial

* El número áureo surge de la división en dos de un segmento guardando las siguientes proporciones: La longitud total ***a+b*** es al segmento más largo ***a***, como ***a*** es al segmento más corto ***b***