Técnicas y Herramientas Modernas

2025-04-23

1.4 Generar un vector secuencia

De hecho, **R** tiene un comando para generar secuencias llamado `seq()`. Recomendamos ejecutar la ayuda del comando en **RStudio**.

Sin embargo, utilizaremos el clásico método de secuencias con anidamiento `for`, `while`, `do`, `until`.

Generaremos una secuencia de números que aumentan de dos en dos

```
# Generamos la secuencia
a = seq(1, 100, 2)

# Mostramos el resultado

a
```

```
##  [1]  1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
## [26] 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95 97 99
```

##1.5 Implementación de una serie Fibonachi o Fibonacci

En matemáticas, la sucesión o serie de Fibonacci es la siguiente sucesión infinita de números naturales: 0,1,1,2,3,5,8 … 89,144,233 … La sucesión comienza con los números 0 y 1. A partir de estos, cada término es la suma de los dos anteriores, lo cual define la relación de recurrencia característica de esta serie.

A los elementos de esta sucesión se los llama números de Fibonacci. Esta sucesión fue descrita en Europa por Leonardo de Pisa, matemático italiano del siglo XIII, también conocido como Fibonacci.

Tiene numerosas aplicaciones en ciencias de la computación, matemáticas y teoría de juegos. También aparece en configuraciones biológicas, como por ejemplo en las ramas de los árboles, en la disposición de las hojas en el tallo, en las flores de alcachofas y girasoles, en las inflorescencias del brócoli romanesco, en la configuración de las piñas de las coníferas, en la reproducción de los conejos y en cómo el ADN codifica el crecimiento de formas orgánicas complejas.

De igual manera, se encuentra en la estructura espiral del caparazón de algunos moluscos, como el nautilus.

Original de la Biblioteca de la Universidad de Florencia. Liber Abaci – Autor: Fibonacci.

```
# Función que genera la sucesión de Fibonacci hasta n términos
fibonacci <- function(n) {
  secuencia <- numeric(n)
  secuencia[1] <- 0
  if (n > 1) {
    secuencia[2] <- 1
    for (i in 3:n) {
      secuencia[i] <- secuencia[i - 1] + secuencia[i - 2]
    }
```

```
  }
  return(secuencia)
}

# Mostrar los primeros 20 números de la serie de Fibonacci
fibonacci(20)
```

```
##  [1]    0    1    1    2    3    5    8   13   21   34   55   89  144  233  377
## [16]  610  987 1597 2584 4181
```

##1.6 Definición matemática recurrente

f0 = 0 (1) f1 = 1 (2) fn+1 = fn + fn−1 (3) Ejemplo inicial del algoritmo

[1] 0 1 1 2

[1] 4

[1] 0 1 1 2 3

[1] 5

[1] 0 1 1 2 3 5

[1] 6

[1] 0 1 1 2 3 5 8

[1] 7

CONSIGNA: ¾Cuantas iteraciones se necesitan para generar un número de la serie mayor que 1.000.000 ?

```
# Sucesión de Fibonacci: ¿cuántas iteraciones para superar 1.000.000?
f <- c(0, 1)
i <- 1

while (tail(f, 1) <= 1e6) {
  cat("[", i, "]", f, "\n")      # Muestra la secuencia actual
  i <- i + 1
  f <- c(f, sum(tail(f, 2)))     # Agrega el siguiente número de Fibonacci
}
```

```
## [ 1 ] 0 1
## [ 2 ] 0 1 1
## [ 3 ] 0 1 1 2
## [ 4 ] 0 1 1 2 3
## [ 5 ] 0 1 1 2 3 5
## [ 6 ] 0 1 1 2 3 5 8
## [ 7 ] 0 1 1 2 3 5 8 13
## [ 8 ] 0 1 1 2 3 5 8 13 21
## [ 9 ] 0 1 1 2 3 5 8 13 21 34
```

```
## [ 10 ] 0 1 1 2 3 5 8 13 21 34 55
## [ 11 ] 0 1 1 2 3 5 8 13 21 34 55 89
## [ 12 ] 0 1 1 2 3 5 8 13 21 34 55 89 144
## [ 13 ] 0 1 1 2 3 5 8 13 21 34 55 89 144 233
## [ 14 ] 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
## [ 15 ] 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610
## [ 16 ] 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
## [ 17 ] 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
## [ 18 ] 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584
## [ 19 ] 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181
## [ 20 ] 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765
## [ 21 ] 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946
## [ 22 ] 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711
## [ 23 ] 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657
## [ 24 ] 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657 46368
## [ 25 ] 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657 46368
## [ 26 ] 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657 46368
## [ 27 ] 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657 46368
## [ 28 ] 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657 46368
## [ 29 ] 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657 46368
## [ 30 ] 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657 46368
```

```
cat("[", i, "]", f, "\n")       # Muestra la secuencia final (mayor a 1 millón)
```

```
## [ 31 ] 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657 46368
```

```
cat("Cantidad de iteraciones necesarias:", i, "\n")
```

```
## Cantidad de iteraciones necesarias: 31
```

###1.7 La Ordenación de burbuja (Bubble Sort en inglés) es un sencillo algoritmo de ordenamiento. Funciona revisando cada elemento de la lista que va a ser ordenada con el siguiente, intercambiándolos de posición si están en el orden equivocado. Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada. Este algoritmo obtiene su nombre de la forma con la que suben por la lista los elementos durante los intercambios, como si fueran pequeñas burbujas. También es conocido como el método del intercambio directo. Dado que solo usa comparaciones para operar elementos, se lo considera un algoritmo de comparación, siendo uno de los más sencillos de implementada.

Tomo una muestra de 10 números ente 1 y 100

x<-sample(1:100,10)

Creo una funci?n para ordenar

burbuja <- function(x){ n<-length(x) for(j in 1:(n-1)){ Programación en R 9 for(i in 1:(n-j)){ if(x[i]>x[i+1]){ temp<-x[i] x[i]<-x[i+1] x[i+1]<-temp } } } return(x) } res<-burbuja(x) #Muestra obtenida x ## [1] 7 71 10 72 37 28 64 82 19 88 #Muestra Ordenada res ## [1] 7 10 19 28 37 64 71 72 82 88 #Ordanaci?n con el coamando SORT de R-Cran sort(x) ## [1] 7 10 19 28 37 64 71 72 82 88

CONSIGNA: Compara la performance de ordenación del método burbuja vs el método sort de R Usar método microbenchmark para una muestra de tamaño 20.000

```
# Cargar librería necesaria
library(microbenchmark)

# Generar muestra aleatoria de 20.000 elementos
muestra <- sample(1:100000, 20000)

# Implementación del algoritmo Bubble Sort
bubble_sort <- function(x) {
  n <- length(x)
  for (j in 1:(n - 1)) {
    for (i in 1:(n - j)) {
      if (x[i] > x[i + 1]) {
        temp <- x[i]
        x[i] <- x[i + 1]
        x[i + 1] <- temp
      }
    }
  }
  return(x)
}

# Comparación de performance con microbenchmark
resultados <- microbenchmark(
  BubbleSort = bubble_sort(muestra),
  Rsort = sort(muestra),
  times = 10
)

# Mostrar resultados
print(resultados)
```

```
## Unit: microseconds
##        expr          min          lq         mean       median           uq
##  BubbleSort 19049765.638 19201928.52 1.959117e+07 19558809.043 19821327.149
##       Rsort      670.831      802.07 8.059447e+02      828.291      840.451
##          max neval
##  20511161.32    10
##       877.68    10
```

###1.8 La penitencia de Newton

Comentan algunos historiadores que Newton tenía en le escuela primaria un profesor muy exigente que reclamaba constantemente la atención de los alumnos a lo que el estaba enseñando. Tan pronto alguien cometía un acto improcedente lo castigaba con una penitencia que consistía en traer para el día siguiente el resultado de la suma de desde el número 1 hasta cierto número aleatoriamente elegido. Por supuesto esta tarea demandaría quedarse toda la noche haciendo las sumas parciales, tarea que el profesor ya había realizado previamente y que tenía registrado en un voluminoso cuaderno que (al esltio tabla de logaritmos) cargaba religiosamente en su portafolios de cuero. #Consigna de trabajo : Desarrolla dos algoritmos que hagan este trabajo , por ejemplo para sumar desde 1 hasta $1 * 1010$ y verifica cuál de los dos es más eficiente

```r
# Algoritmo 1: Suma con bucle for
suma_for <- function(n) {
  suma <- 0
  for (i in 1:n) {
    suma <- suma + i
  }
  return(suma)
}

# Algoritmo 2: Suma con fórmula de Gauss
suma_gauss <- function(n) {
  return(n * (n + 1) / 2)
}

# Valor elegido para n (100 millones)
n <- 1e8

# Ejecutamos ambos métodos
resultado_for <- suma_for(n)
resultado_gauss <- suma_gauss(n)

# Mostramos los resultados
resultado_for
```

```
## [1] 5e+15
```

```r
resultado_gauss
```

```
## [1] 5e+15
```