Módulo II - Programación R-Cran

Pablo Martino

pablomartino94@gmail.com

Abstract. En este documento documentamos los ejercicios realizados como actividad para el módulo 2 sacados del pdf:

https://themys.sid.uncu.edu.ar/rpalma/TyHM/Benchmark/Elementos-de-Programaci%c3%b3n.pdf

Keywords: Programing Language · R-Cran · RMarkdown

RESOLUCIONES DE LOS EJERCICIOS DEL TRABAJO PRACTICO:

EJERCICIO 1: Generación de vector secuencia

Se generó una secuencia de números entre 1 y 1.000.000 que va de dos en dos, con el comando for y con el comando ”seq” de R. Luego, medimos cuál de los dos comandos ejecuta la instrucción más rápido utilizando el método: Sys.time.

1.1. Secuencia generada con for:

```
> A<-0
> start_time<-Sys.time()
> for (i in 1:50000) { A[i] <- (i*2)}
> head(A)
> tail(A)
> end_time<-Sys.time()
> end_time-start_time
```

RESULTADOS

```
> A<-0
> start_time<-Sys.time()
> for (i in 1:50000) {A[i]<-(i*2)
+
+ }
> head(A)
[1]  2  4  6  8 10 12
> tail(A)
[1]  99990  99992  99994  99996  99998 100000
> end_time<-Sys.time()
> end_time - start_time

Time difference of 0.02649713 secs
```

1.2. Secuencia generada con ”seq”:

```
> start_time <- Sys.time()
> A <- seq(1,100000,2)
> head (A)
> tail(A)
> end_time <- Sys.time()
> end_time - start_time
```

RESULTADOS

```
> start_time<-Sys.time()
> A<-seq(1,100000,2)
> head(A)
[1]  1  3  5  7  9 11
> tail(A)
[1] 99989 99991 99993 99995 99997 99999
> end_time<-Sys.time()
> end_time-start_time

Time difference of 0.0009310246 secs
```

Conclusión: Pudimos observar que el comando ”seq” de R genera la secuencia de números más rápido.

EJERCICIO 2: Implementación de una serie Fibonachi o Fibonacci

Se generó una secuencia de números, donde cada término de la secuencia es la suma de los dos términos anteriores.
0,1,1,2,3,5,8 ... 89,144,233 …
f0 = 0;
f1 = 1;
fn+1 = fn + fn−1

Luego, buscaremos la cantidad de interacciones realizadas en R para responder la pregunta ¿Cuántas iteraciones se necesitan para generar un número de la serie mayor que 1.000.000?

```
> f0<-0
> f1<-1
> it<-0
> f2<-0
> S<-c(f0,f1)
> while(f2<=1000000){
+     it<-(it+1)
+     f2<-(f0+f1)
+     S<-c(S,f2)
+     f0<-f1
+     f1<-f2
+ }
> it

[1] 30
```

Conclusión: Se necesitan 30 iteraciones para generar un número mayor que 1.000.000 siguiendo la serie de Fibonacci.

EJERCICIO 3: Ordenación de un vector por método burbuja

Se generó una serie de números aleatoria entre el 0 y el 100 y la ordeno con el comando burbuja y comando ”sort” de R. Luego comparo tiempos de ejecución con una muestra de 20.000 con el método Microbenchmark.

3.1 Método Burbuja

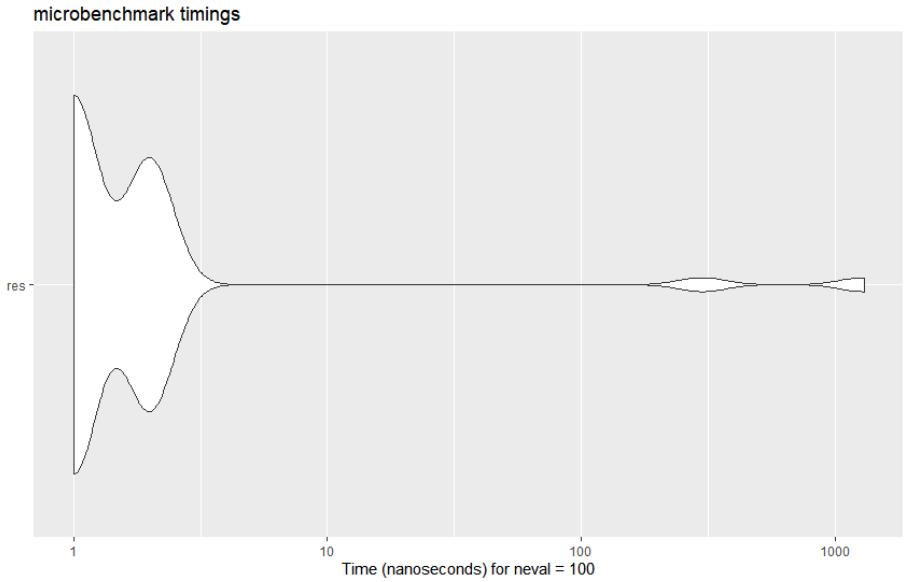
```
> library(microbenchmark)
> # Tomo una muestra de 10 números ente 1 y 20000
> x<-sample(1:20000,10)
> starttime<-Sys.time()
> # Creo una funci?n para ordenar
> burbuja <- function(x){
+     n<-length(x)
+     for(j in 1:(n-1)){
+         for(i in 1:(n-j)){
+             if(x[i]>x[i+1]){
+                 temp<-x[i]
+                 x[i]<-x[i+1]
+                 x[i+1]<-temp
+             }
+         }
+     }
+     return(x)
+ }
> res<-burbuja(x)
> mbm<-microbenchmark(res)
> endtime<-Sys.time()
> mbm
> endtime-starttime
> autoplot(mbm)
```

RESULTADOS

```
> library(microbenchmark)
> # Tomo una muestra de 10 números ente 1 y 20000
> x<-sample(1:20000,10)
> starttime<-Sys.time()
> # Creo una funci?n para ordenar
> burbuja <- 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)
+ }
> res<-burbuja(x)
> mbm<-microbenchmark(res)
Aviso:
In microbenchmark(res) :
  Could not measure a positive execution time for 39
evaluations.
> endtime<-Sys.time()
> mbm
Unit: nanoseconds
 expr min lq  mean median uq  max neval
  res   0  0 16.65      0  1 1301   100
> endtime-starttime
Time difference of 0.1821139 secs
> autoplot(mbm)
Avisos:
1: In ggplot2::scale_y_log10(name = y_label) :
  log-10 transformation introduced infinite values.
2: Removed 53 rows containing non-finite outside the
scale range (`stat_ydensity()`).
```

Plot



3.2 Con comando ”sort”

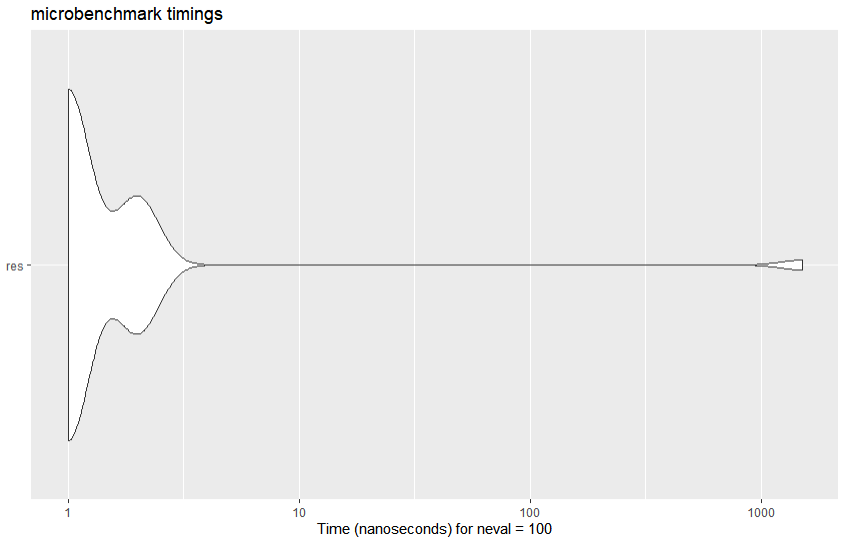
```
> library(microbenchmark)
> # Tomo una muestra de 10 números ente 1 y 20000
> x<-sample(1:20000,10)
> starttime<-Sys.time()
> # Creo una funci?n para ordenar
> a<-sort(x)
> mbm<-microbenchmark(res)
> endtime<-Sys.time()
> mbm
> endtime-starttime
> autoplot(mbm)
```

RESULTADOS

```
> library(microbenchmark)
> # Tomo una muestra de 10 números ente 1 y 20000
> x<-sample(1:20000,10)
> starttime<-Sys.time()
> # Creo una funci?n para ordenar
> a<-sort(x)
> mbm<-microbenchmark(a)
Aviso:
```

```
In microbenchmark(a) :
  Could not measure a positive execution time for 15
evaluations.
> endtime<-Sys.time()
> mbm
Unit: nanoseconds
 expr min lq  mean median uq  max neval
    a   0  0 17.79      1  1 1701   100
> endtime-starttime
Time difference of 0.1985579 secs
> autoplot(mbm)
Avisos:
1: In ggplot2::scale_y_log10(name = y_label) :
  log-10 transformation introduced infinite values.
2:  Removed  37  rows  containing  non-finite  outside  the
scale range (`stat_ydensity()`).
```

Plot



Conclusión: A través de las gráficas y datos, podemos observar que el método burbuja es más rápido y que requiere menos recursos para procesar el comando.