Programacion en R

Elias Emmanuel da ROSA[*,a], Daniela Belen GENOVESE[*,a], Agustin Cobos[*,a], Facundo Fortes del CAMPO[*,a], Laura MENECES[*,a], Julian SIBECAS[*,a]

[a] *Facultad de Ingeniería - Universidad Nacional de Cuyo, Boulogne Sur Mer 683, Mendoza, Argentina*

Abstract

En este breve tutorial examinaremos algunos elementos del lenguaje de programacion R y como valernos de ello para resolver problemas de la vida cotidiana. Apelaremos a ejemplos bien conocidos, pero además mostraremos las soluciones que desarrollaremos contra las misma que ya están implementadas en R. Comparando el **costo computacional**, medido como tiempo de ejecución. Esto nos permitirá entender la calidad del algoritmo que implementemos. Como excusa para introducirnos propondremos realizar tres experimentos y medir el tiempo ejecución. Veremos:

- Generar un vector secuencia
- Implementación de una serie Fibonacci
- Ordenación de un vector por método burbuja

Generar un vector secuencia

De echo R. tiene un comando para generar secuencias llamado "seq." Recomendamos ejecutar la ayuda del comando en RStudio. Pero utilizaremos el clásico método de secuencias de anidamiento for, while, do , until. Generaremos una secuencia de números que de dos en dos entre 1 y 100.000.

Consigna. Comparar la performance con systime

Resolución. El algoritmo es el siguiente:

```
start_time_r <- Sys.time()
R_seq <- seq(1,1000000, 2)
end_time_r <- Sys.time()
performance_r <- end_time_r - start_time_r
print(performance_r)
```

*Todos los integrantes, son del grupo Ladef.

Email addresses: eliasdarosa99@hotmail.com (Elias Emmanuel da ROSA), danibelgenovese@gmail.com (Daniela Belen GENOVESE), agustinscobos@gmail.com (Agustin Cobos), facu.fortes.96@gmail.com (Facundo Fortes del CAMPO), lauramenecesa81@gmail.com (Laura MENECES), jsibecas@gmail.com (Julian SIBECAS)

```
## Time difference of 0.03100204 secs
```

```
For_seq <- ""
start_time_for <- Sys.time()
for (i in 1:50000) { For_seq[i] <- (i*2)}
end_time_for <- Sys.time()
performance_for <- end_time_for - start_time_for
print(performance_for)
```

```
## Time difference of 0.3000169 secs
```

```
ifelse(performance_r > performance_for, "for was faster" , "R was faster")
```

```
## [1] "R was faster"
```

Conclusión. De ambos metodos, el más rápido (y por lo tanto, con mejor performance) es generar una secuencia por medio de seq de R

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 1,2 a partir de estos, «cada término es la suma de los dos anteriores», es la relación de recurrencia que la define.

A los elementos de esta sucesión se les 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, matemática 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 brécol romanesco, en la configuración de las piñas de las coníferas, en la reproducción de los conejos y en como 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.

Consigna. ¿Cuántas iteraciones se necesitan para generar un número de la serie mayor que 1.000.000 ?

Resolución. El algoritmo es el siguiente:

```
n1 <- 0
n2 <- 1
count = 0
nth <- n1+n2
    while(nth < 1000000 ) {
```

```
      nth <- n1 + n2
      n1 <- n2
      n2 <- nth
      count <- count + 1

    }
print("El numero de iteraciones es: " )
```

```
## [1] "El numero de iteraciones es: "
```

```
      print(count)
```

```
## [1] 30
```

```
      print("El valor de la variable es: " )
```

```
## [1] "El valor de la variable es: "
```

```
      print(nth)
```

```
## [1] 1346269
```

Conclusión. Se requirieron 30 iteraciones para generar un número de la serie mayor a 1.000.000

Ordenación de un vector por método burbuja

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.

Consigna. Comparar la performance de ordenación del método burbuja vs el método sort de R. Usar método microbenchmark para una muestra de tamaño 20.000.

Aclaración. Se modificó el límite, y en vez de hacer 20000 se hizo 2000, por un tema de rapidez en el uso de la PC. Con 20000 tardaba mucho tiempo el algoritmo, no así con 2000.

Resolución. El algoritmo es el siguiente:

```
print("Inicia ordenamiento por metodo burbuja " )
```

```
## [1] "Inicia ordenamiento por metodo burbuja "
```

```
# Tomo una muestra de 10 números ente 1 y 100
x<-sample(1:100000,2000)

tburbujai<-Sys.time()
# Creo una funcion 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)
tburbujaf<-Sys.time()
tiempo1<-tburbujaf-tburbujai
print("El tiempo total del metodo de burbuja es: " )
```

```
## [1] "El tiempo total del metodo de burbuja es: "
```

```
print (tiempo1)
```

```
## Time difference of 0.409023 secs
```

```
print("Inicia ordenamiento por sort " )
```

```
## [1] "Inicia ordenamiento por sort "
```

```
tsorti<-Sys.time()
or<-sort (x)
tsortf<-Sys.time()
tiempo2<-tsortf-tsorti
print("El tiempo total del metodo sort es:")
```

```
## [1] "El tiempo total del metodo sort es:"
```

```
print(tiempo2)
```

```
## Time difference of 0.003000975 secs
```

```
if (tiempo1>tiempo2)
  print ("El que tarda menos tiempo es sort")
```

```
## [1] "El que tarda menos tiempo es sort"
```

```
if (tiempo2>tiempo1)
  print ("El que tarda menos tiempo es burbuja")
```

```
library(microbenchmark)
```

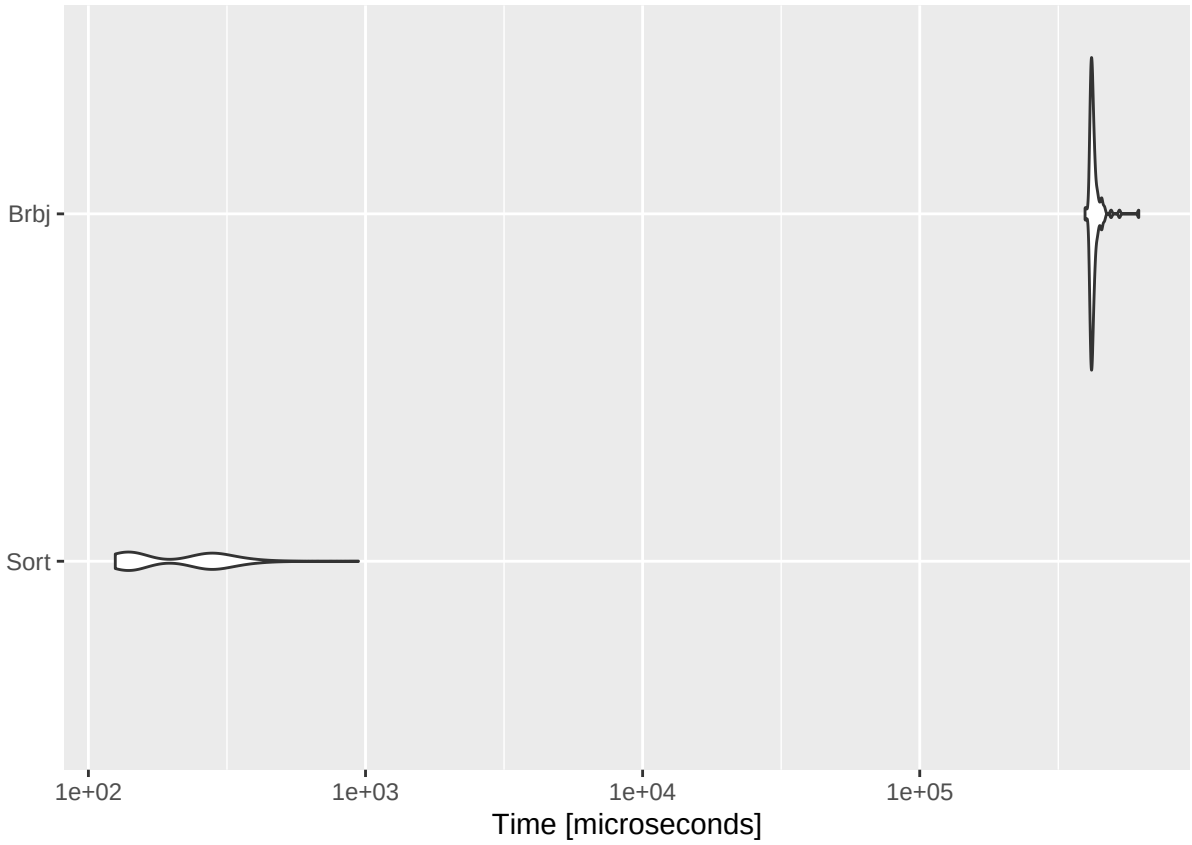
```
otro<-microbenchmark("Sort" = a <- sort(x),
                     "Brbj"= b <- burbuja(x))

otro
```

```
## Unit: microseconds
##  expr        min         lq        mean     median          uq        max neval
##  Sort    125.048    140.298    229.2342    249.818    282.8135    942.988   100
##  Brbj 394873.762 414556.035 426031.1636 419323.240 427696.7280 617170.754   100
```

```
library(ggplot2)
autoplot(otro)
```

```
## Coordinate system already present. Adding new coordinate system, which will replace the e
```



Conclusión. De ambos métodos, el más rápido (y por lo tanto, con mejor performance) es ordenar por medio del sort de R. También podemos ver que systime y microbenchmark practicamente dicen lo mismo, pero el microbenchmark es mas exacto, y detallado.