Técnicas y Herramientas Modernas I- Introducción a lenguajes de programación de alto nivel (paradigma de objetos) – Programación en R

Futuros Ingenieros

2024-03-27

Table of the cost of ownership

Elementos	Costo
Servidor	1500999
Computadoras	1265000
Aire Acondicionado	780000

Comandos

Subtitulos

Para crear un subtitulo se ponen dos numerales antes de la palabra

Links

Para agregar un URL que te dirija a la página hay que ponerlo entre <>

Negrita

Para resaltar las palabras en negrita se utilizan ** antes y ** despues de las palabras

Gráfico

Para graficar se debe poner “plot()” y entre los parentesis lo que se desea graficar

Código {r pressure, echo=FALSE} plot(pressure)

Cuadro

Para crear cuadro se deben usar barras y guiones con el formato de cuadro como se muestra a continuación

Columna 1	Columna 2
Fila	Contenido
Fila 2	Contenido
Fila 3	Contenido

Vector

```
mi_vector_a <- c(12,34,12,54,23,12,65,34,12,56,66)

mi_vector_b <- seq(1:16)

print(mi_vector_a)
```

```
##  [1] 12 34 12 54 23 12 65 34 12 56 66
```

```
print(mi_vector_b)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
```

Matrices

Las matrices tienen filas y numeros. Se alimentan de vectores. Solo son números.

```
mi_matriz_c <- matrix(mi_vector_b,nrow=4, byrow=TRUE)
print(mi_matriz_c)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
## [4,]   13   14   15   16
```

Cómo traer un elemento de la matriz

Uso las filas y columnas entre corchetes

```
mi_matriz_c[4,2]
```

```
## [1] 14
```

Cómo traer la fila 4 completa

```
mi_matriz_c[4,]
```

```
## [1] 13 14 15 16
```

Cómo trer una columna

```
mi_matriz_c[,1]
```

```
## [1]  1  5  9 13
```

Cómo eliminar una fila

```
 mi_matriz_c[-2,]
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    9   10   11   12
## [3,]   13   14   15   16
```

EJERCICIO N 1: Penitencia de Gauss

Sys.time

Usando Sys.time el tiempo de ejecución de un fragmento de código se puede medir tomando la diferencia entre el tiempo al inicio y al final del fragmento de código leyendo los registros del RTC (Real Time Clock.

Simple pero flexible, como un relojito de arena :

```
sleep_for_a_minute <- function(){Sys.sleep(14)}
start_time <- Sys.time()
sleep_for_a_minute()
end_time <- Sys.time()

end_time - start_time
```

```
## Time difference of 14.01552 secs
```

Hemos generado una función que antes no existía y la hemos usado. Si usas el comando dentro de un documento en R-Studio te demorarás mucho tiempo cuando compiles un PDF o una presentación.

Ahora se compararon dos códigos que tienen la misma finalidad, con el fin de observar su rendimiento y eficiencia. Ambos códigos sirven para generar un vector con una secuencia numérica.

##Codigo generado con for:

```
A<- c()
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 0.0629034 secs
```

##Codigo generado con R

```
start_time2<- 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_time2 <- Sys.time()
end_time2 - start_time2
```

```
## Time difference of 0.03705192 secs
```

EJERCICIO N 2: Implementacion de serie de 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. . . 00 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.

$$f_0 = 0; f_1 = 1; f_{n+1} = f_n + f_{n-1}$$

Como consigna de este ejercicio se debe buscar cuantas iteraciones se deben realizar para que el valor de la serie alcance un número mayor a 1.000.000

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

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

```
tail(vec)
```

```
## [1]  121393  196418  317811  514229  832040 1346269
```

Se puede observar que el algoritmo requiere al menos 30 iteraciones para poder superar el millón.

EJERCICIO N 3:Ordenamiento de un vector por el método burbuja

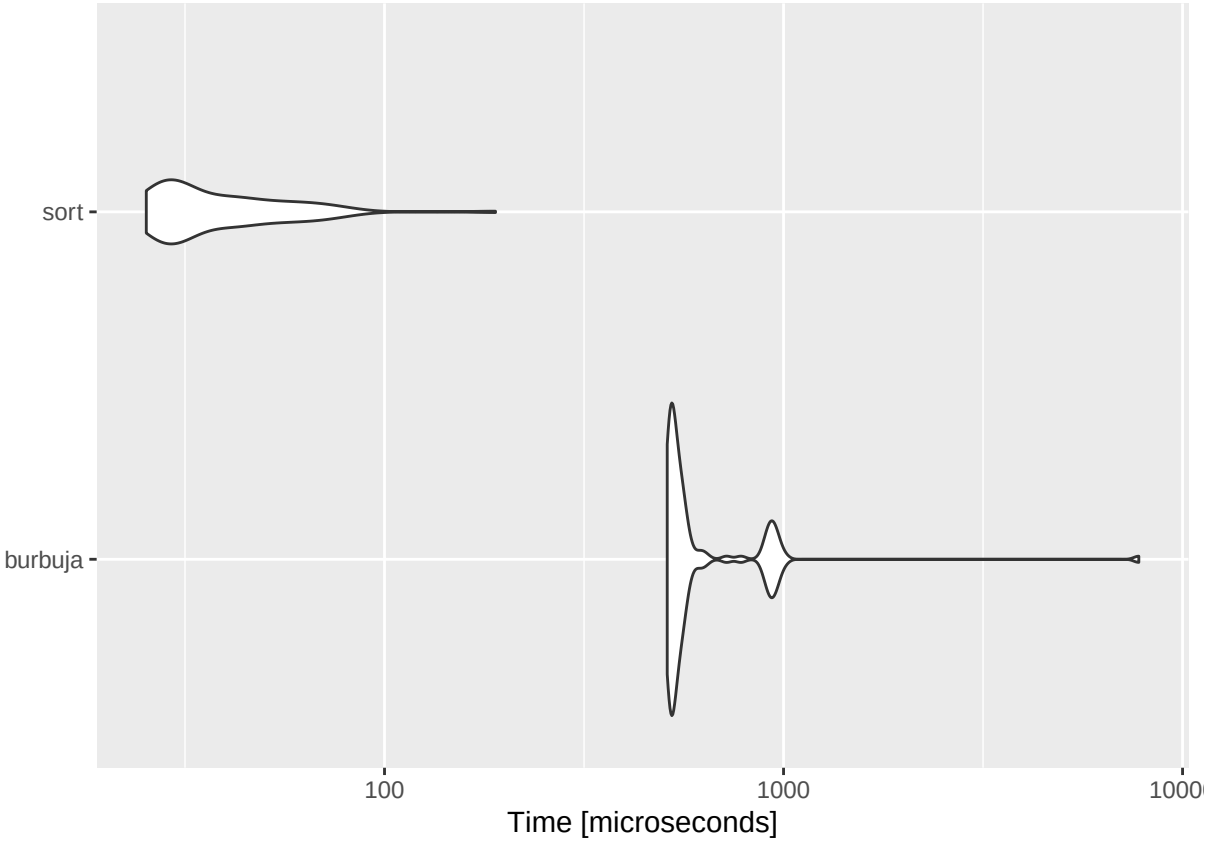
En el siguiente ejercicio se realiza el ordenamiento de los valores numéricos de un vector mediante el método burbuja vs. el método sort nativo de R.

```
library(microbenchmark)
x<-sample(1:100,100)
mbm<-microbenchmark(
##método de ordenamiento directo o burbuja
"burbuja"={
  burbuja<-function(x){
    n<-length(x)
    for(j in 1:(n-1)){
      for(i in 1:(n-j)){
        if(x[i]>x[i+1]){
          temporal<-x[i]
          x[i]<-x[i+1]
          x[i+1]<-temporal
        }
      }
    }
  return(x)
  }
res<-burbuja(x)
},
##método de R sort
"sort"={
  sort(x)
}
)
mbm
```

```
## Unit: microseconds
##     expr    min       lq      mean  median       uq      max neval
##  burbuja 511.10 521.2305 686.35749 539.465 584.0700 7769.746   100
```

```
##     sort  25.29  28.3900  41.26243  32.575  47.2805  189.720   100
```

```
library(ggplot2)
autoplot(mbm)
```



CONSIGNA: Compara la performance de ordenacióndel método burbuja vs el método sort de R.

Usar método microbenchmark para una muestra de tamaño 20.000.

Realizamos la comparativa atraves de esta graf.