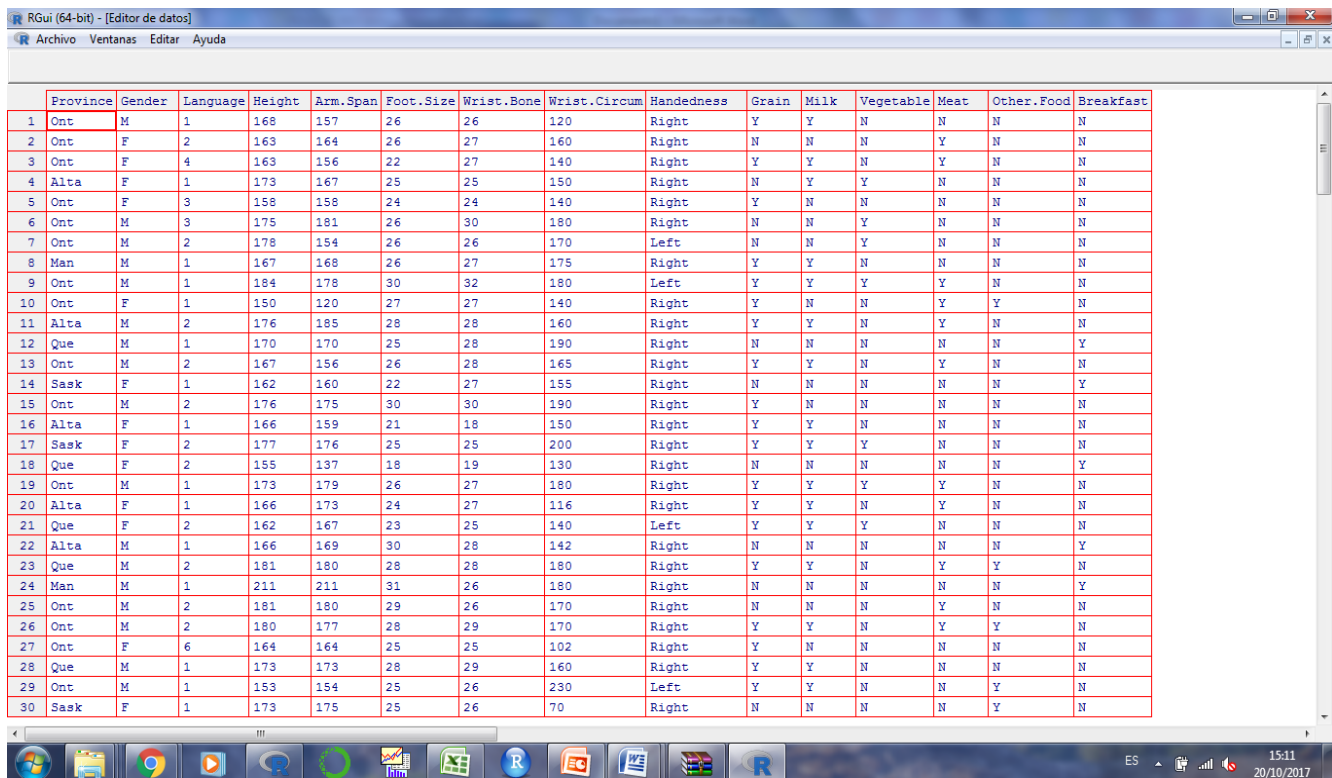


Ejercicio 1. Búsqueda y Selección.

Cargar los datos de “student_census.csv” del repositorio haciendo uso de read.csv. Para una celebración deportiva en el campus, se quiere saber si es posible formar un equipo de 5 estudiantes que jueguen al Baloncesto. Para ello se pide localizar a todos los estudiantes que practiquen dicho deporte y de ellos seleccionar 5 mediante una selección aleatoria. Para seleccionarlos se implementará una función que genera números aleatorios uno a uno en el rango adecuado, si el estudiante aún no forma parte del equipo se integrará y si ya está se generará otro número aleatorio hasta que estén seleccionados los cinco.

- Se cargan los datos.(Introduciendo fix(datos), aparece la siguiente ventana:



| | Province | Gender | Language | Height | Arm.Span | Foot.Size | Wrist.Bone | Wrist.Circum | Handedness | Grain | Milk | Vegetable | Meat | Other.Food | Breakfast |
|----|----------|--------|----------|--------|----------|-----------|------------|--------------|------------|-------|------|-----------|------|------------|-----------|
| 1 | Ont | M | 1 | 168 | 157 | 26 | 26 | 120 | Right | Y | Y | N | N | N | N |
| 2 | Ont | F | 2 | 163 | 164 | 26 | 27 | 160 | Right | N | N | N | Y | N | N |
| 3 | Ont | F | 4 | 163 | 156 | 22 | 27 | 140 | Right | Y | Y | N | Y | N | N |
| 4 | Alta | F | 1 | 173 | 167 | 25 | 25 | 150 | Right | N | Y | Y | N | N | N |
| 5 | Ont | F | 3 | 158 | 158 | 24 | 24 | 140 | Right | Y | N | N | N | N | N |
| 6 | Ont | M | 3 | 175 | 181 | 26 | 30 | 180 | Right | N | N | Y | N | N | N |
| 7 | Ont | M | 2 | 178 | 154 | 26 | 26 | 170 | Left | N | N | Y | N | N | N |
| 8 | Man | M | 1 | 167 | 168 | 26 | 27 | 175 | Right | Y | Y | N | N | N | N |
| 9 | Ont | M | 1 | 184 | 178 | 30 | 32 | 180 | Left | Y | Y | Y | Y | N | N |
| 10 | Ont | F | 1 | 150 | 120 | 27 | 27 | 140 | Right | Y | N | N | Y | Y | N |
| 11 | Alta | M | 2 | 176 | 185 | 28 | 28 | 160 | Right | Y | Y | N | Y | N | N |
| 12 | Que | M | 1 | 170 | 170 | 25 | 28 | 190 | Right | N | N | N | N | N | Y |
| 13 | Ont | M | 2 | 167 | 156 | 26 | 28 | 165 | Right | Y | Y | N | Y | N | N |
| 14 | Sask | F | 1 | 162 | 160 | 22 | 27 | 155 | Right | N | N | N | N | N | Y |
| 15 | Ont | M | 2 | 176 | 175 | 30 | 30 | 190 | Right | Y | N | N | N | N | N |
| 16 | Alta | F | 1 | 166 | 159 | 21 | 18 | 150 | Right | Y | Y | N | N | N | N |
| 17 | Sask | F | 2 | 177 | 176 | 25 | 25 | 200 | Right | Y | Y | Y | N | N | N |
| 18 | Que | F | 2 | 155 | 137 | 18 | 19 | 130 | Right | N | N | N | N | N | Y |
| 19 | Ont | M | 1 | 173 | 179 | 26 | 27 | 180 | Right | Y | Y | Y | Y | N | N |
| 20 | Alta | F | 1 | 166 | 173 | 24 | 27 | 116 | Right | Y | Y | N | Y | N | N |
| 21 | Que | F | 2 | 162 | 167 | 23 | 25 | 140 | Left | Y | Y | Y | N | N | N |
| 22 | Alta | M | 1 | 166 | 169 | 30 | 28 | 142 | Right | N | N | N | N | N | Y |
| 23 | Que | M | 2 | 181 | 180 | 28 | 28 | 180 | Right | Y | Y | N | Y | Y | N |
| 24 | Man | M | 1 | 211 | 211 | 31 | 26 | 180 | Right | N | N | N | N | N | Y |
| 25 | Ont | M | 2 | 181 | 180 | 29 | 26 | 170 | Right | N | N | N | Y | N | N |
| 26 | Ont | M | 2 | 180 | 177 | 28 | 29 | 170 | Right | Y | Y | N | Y | Y | N |
| 27 | Ont | F | 6 | 164 | 164 | 25 | 25 | 102 | Right | Y | N | N | N | N | N |
| 28 | Que | M | 1 | 173 | 173 | 28 | 29 | 160 | Right | Y | Y | N | N | N | N |
| 29 | Ont | M | 1 | 153 | 154 | 25 | 26 | 230 | Left | Y | Y | N | N | Y | N |
| 30 | Sask | F | 1 | 173 | 175 | 25 | 26 | 70 | Right | N | N | N | N | Y | N |

- Creo un vector vacío (que he llamado lista en el programa) donde se filtrarán los estudiantes, identificados por su posición, que practican baloncesto.

El esquema que se va a seguir es:

- Se crea una lista vacía para almacenar las posiciones (identificador de cada estudiante) que cumple la condición de que jueguen al baloncesto
- se crea un contador, inicializado en 1. En los sucesivos pasos servirá para ir sumando 1 a las posiciones de la lista definida en el punto anterior. Es necesario definirlo, ya que la i del bucle avanza más posiciones que los índices de la lista (contador).
- Se crea un bucle for que recorra todas las filas del data set de la columna physical (que en éste caso he denominado deportes)
 - Si deporte[i]==Basketball----> lista[contador]=i

- Por último, una vez ya filtrados los estudiantes cuyo deporte es el baloncesto en la lista, se toma una muestra aleatoria con la función `sample(lista,5)`

```

R Console
+ }
> busca(0,100, 100)
[1] 9 16
[1] 18 22
[1] 100 100
> deporte<- datos$Physical
> ###PRACTICA 5
> #####
>
> #Ejercicio 1. Búsqueda y Selección.
> #-----
>
> #Cargar los datos de "student_census.csv" del repositorio haciendo uso de read.csv
> #deportiva en el campus, se quiere saber si es posible formar un equipo de 5 $
> #Baloncesto. Para ello se pide localizar a todos los estudiantes que practiquen
> #seleccionar 5 mediante una selección aleatoria. Para seleccionarlos se implementa
> #números aleatorios uno a uno en el rango adecuado, si el estudiante aún no forma
> #y si ya está se generará otro número aleatorio hasta que estén seleccionados 5
>
> datos<- read.csv("C:/Users/usuario/Desktop/programacion R/student_census.csv")
> fix(datos)
> deporte<- datos$Physical
> a<- dim(datos)
> lista<- vector()
> contador<-1
> for (i in 1:a[1]){
+ if (deporte[i]=="Basketball"){
+ lista[contador] = i
+ contador = contador+1}}
> lista
[1] 11 16 18 36 89 92 96 98 103 107 113 132 141 148 157 176 187 200
> ##seleccionar a 5 alumnos de la lista de forma aleatoria
> jugadores<- sample(lista, 5)
> jugadores
[1] 157 113 36 96 132
>

C:\Users\usuario\Desktop\programacion R\nuevop5.R - Editor R
###PRACTICA 5
#####

#Ejercicio 1. Búsqueda y Selección.
#-----

#Cargar los datos de "student_census.csv" del repositorio haciendo uso de read.csv
#deportiva en el campus, se quiere saber si es posible formar un equipo de 5 $
#Baloncesto. Para ello se pide localizar a todos los estudiantes que practiquen
#seleccionar 5 mediante una selección aleatoria. Para seleccionarlos se implementa
#números aleatorios uno a uno en el rango adecuado, si el estudiante aún no forma
#y si ya está se generará otro número aleatorio hasta que estén seleccionados 5

datos<- read.csv("C:/Users/usuario/Desktop/programacion R/student_census.csv")
fix(datos)
deporte<- datos$Physical
a<- dim(datos)
lista<- vector()
contador<-1
for (i in 1:a[1]){
  if (deporte[i]=="Basketball"){
    lista[contador] = i
    contador = contador+1}}

lista
##seleccionar a 5 alumnos de la lista de forma aleatoria
jugadores<- sample(lista, 5)
jugadores

#Ejercicio 2. Ordenación y Regresión.
#-----

#Con los datos de "student_census.csv" del ejercicio anterior, se pide filtrar
#Arm.Span, Foot.Size y realizar correlaciones dos a dos. Ordenar los datos de
#determinar la procedencia de los alumnos más altos y hacer un diagrama de barras
#por provincias.

```

Ejercicio 2. Ordenación y Regresión.

Con los datos de "student_census.csv" del ejercicio anterior, se pide filtrar en un data.frame las variables Height, Arm.Span, Foot.Size y realizar correlaciones dos a dos. Ordenar los datos de acuerdo con la estatura para determinar la procedencia de los alumnos más altos y hacer un diagrama de barras que indique los resultados por provincias.

- Creo un dataframe llamado matriz formado por las tres columnas del dataset: Height, Arm.Span, Foot.Size.
- Hago la matriz de correlaciones de la matriz (`cor(matriz)`): se obtiene una matriz simétrica con la diagonal de unos. Ésto es porque la matriz de correlaciones calcula la correlación de cada variable con el resto y con ella misma.
- Utilizo el comando `order` para que me devuelva cada índice de las posiciones según la ordenación de la primera variable introducida el `order`, en éste caso `height`.
- Utilizo el comando `cbind` para ordenar los valores de cada variable según los índices ordenados con el comando `order`.

- Por último, se utiliza el comando plot para realizar un diagrama de barras con la frecuencia de cada provincia.

The screenshot shows the RGui (32-bit) interface. The main window is the R script editor, which contains the following code:

```
#números aleatorios uno a uno en el rango adecuado, si el estudiante aún no for
#y si ya está se generará otro número aleatorio hasta que estén seleccionados 1

datos<- read.csv("C:/Users/usuario/Desktop/programacion R/student_census.csv",
fix(datos)
deporte<- datos[,16]
a<- dim(datos)
lista<- vector()
contador<-1
for (i in 1:a[1]){
  if (datos[i,16]=="Basketball"){
    lista[contador] = i
    contador = contador+1}}
lista
##seleccionar a 5 alumnos de la lista de forma aleatoria
jugadores<- sample(lista, 5)

#Ejercicio 2. Ordenación y Regresión.
#-----

#Con los datos de "student_census.csv" del ejercicio anterior, se pide filtrar
#Arm.Span, Foot.Size y realizar correlaciones dos a dos. Ordenar los datos de
#determinar la procedencia de los alumnos más altos y hacer un diagrama de bar
#por provincias.

datos<- data.frame(datos)
height<- data.frame(datos$Height)
arm.span<- data.frame(datos$Arm.Span)
foot.size<- data.frame(datos$Foot.Size)
matriz<- data.frame(height, arm.span, foot.size)

## matriz de correlacion
cor(matriz)

##te devuelve el indice de las posiciones ordenadas
(ii<- order(height, arm.span, foot.size))
cbind(height, arm.span, foot.size)[ii,]
```

The console window shows the output of the correlation matrix calculation:

```
> ## matriz de correlacion
> cor(matriz)
      datos.Height datos.Arm.Span datos.Foot.Size
datos.Height      1.0000000      0.7498349      0.5006804
datos.Arm.Span      0.7498349      1.0000000      0.4615200
datos.Foot.Size      0.5006804      0.4615200      1.0000000
> |
```

```
RGui (64-bit)
Archivo Editar Visualizar Misc Paquetes Ventanas Ayuda

R R Console
> #por provincias.
>
> datos<- data.frame(datos)
> height<- data.frame(datos$Height)
> arm.span<- data.frame(datos$Arm.Span)
> foot.size<- data.frame(datos$Foot.Size)
> matriz<- data.frame(height, arm.span, foot.size)
>
> ## matriz de correlacion
> cor(matriz)
      datos.Height datos.Arm.Span datos.Foot.Size
datos.Height      1.0000000      0.7498349      0.5006804
datos.Arm.Span      0.7498349      1.0000000      0.4615200
datos.Foot.Size      0.5006804      0.4615200      1.0000000
>
> ##te devuelve el indice de las posiciones ordenadas
> (ii<- order(height, arm.span, foot.size))
[1] 113 183 144 179 66 10 64 71 154 62 194 63 169 29 139 115 100 197
[19] 18 163 89 88 48 153 51 79 171 140 50 132 5 192 164 195 134 142
[37] 123 116 186 77 173 148 157 128 182 178 92 96 31 172 189 146 137 68
[55] 159 14 161 37 90 21 49 136 162 3 170 147 85 126 130 2 44 45
[73] 124 36 27 38 119 81 82 188 86 166 42 84 117 16 93 112 175 22
[91] 20 99 13 168 101 43 83 111 8 55 107 143 125 60 1 34 190 104
[109] 177 155 110 131 69 180 114 40 33 91 106 12 61 32 94 35 58 80
[127] 185 181 156 72 184 149 54 102 151 109 108 76 87 4 28 158 30 19
[145] 145 167 152 41 196 105 176 73 6 141 122 121 15 46 11 200 127 17
[163] 7 97 57 120 75 193 118 95 26 150 135 129 78 198 23 25 138 74
[181] 53 98 70 47 9 67 103 133 165 39 199 160 65 191 187 174 52 56
[199] 59 24
> cbind(height, arm.span, foot.size)[ii,]
      datos.Height datos.Arm.Span datos.Foot.Size
113          126          150          24
183          128          157          23
144          130          125          19
179          140          140          22
66           147          144          19

R C:\Users\usuario\Desktop\programacion R\nuevop5.R- Editor R
##seleccionar a 5 alumnos de la lista de forma aleatoria
jugadores<- sample(lista, 5)
jugadores

#Ejercicio 2. Ordenación y Regresión.
#-----

#Con los datos de "student_census.csv" del ejercicio anterior, se pide filtrar
#Arm.Span, Foot.Size y realizar correlaciones dos a dos. Ordenar los datos de a
#determinar la procedencia de los alumnos más altos y hacer un diagrama de barr
#por provincias.

datos<- data.frame(datos)
height<- data.frame(datos$Height)
arm.span<- data.frame(datos$Arm.Span)
foot.size<- data.frame(datos$Foot.Size)
matriz<- data.frame(height, arm.span, foot.size)

## matriz de correlacion
cor(matriz)

##te devuelve el indice de las posiciones ordenadas
(ii<- order(height, arm.span, foot.size))
cbind(height, arm.span, foot.size)[ii,]
plot(datos$Province)

#Ejercicio 3. MapReduce. Simulación
#-----

#Con los datos de "student_census.csv" del ejercicio anterior, se pide:
#Particionar el dataset en 20 subconjuntos.
#Map: Ordenar cada uno de los subconjuntos por estatura en orden decreciente us
#Implementar una función que mezcle ordenadamente dos subconjuntos previamente
```

```
RGui (64-bit)
Archivo Editar Paquetes Ventanas Ayuda

R R Console
75      178      200      18
193      179      150      28
118      179      179      22
95      179      183      26
26      180      177      28
150      180      179      27
135      180      181      27
129      180      182      30
78      180      185      25
198      180      188      28
23      181      180      28
25      181      180      29
138      181      181      26
74      181      190      30
53      182      185      32
98      183      173      28
70      183      185      25
47      183      185      27
9      184      178      30
67      184      186      27
103      185      170      27
133      185      181      28
165      185      190      28
39      185      194      30
199      187      182      27
160      187      195      28
65      189      189      30
191      190      160      22
187      190      187      28
174      191      183      24
52      201      201      22
56      203      166      24
59      203      200      31
24      211      211      31
> plot(datos$Province)
>

R C:\Users\usuario\Desktop\programacion R\nuevop5.R- Editor R
contador = contador+1})
lista
##seleccionar a 5 alumnos de la lista de forma aleatoria
jugadores<- sample(lista, 5)
jugadores

#Ejercicio 2. Ordenación y Regresión.
#-----

#Con los datos de "student_census.csv" del ejercicio anterior, se pide filtrar
#Arm.Span, Foot.Size y realizar correlaciones dos a dos. Ordenar los datos de a
#determinar la procedencia de los alumnos más altos y hacer un diagrama de barr
#por provincias.

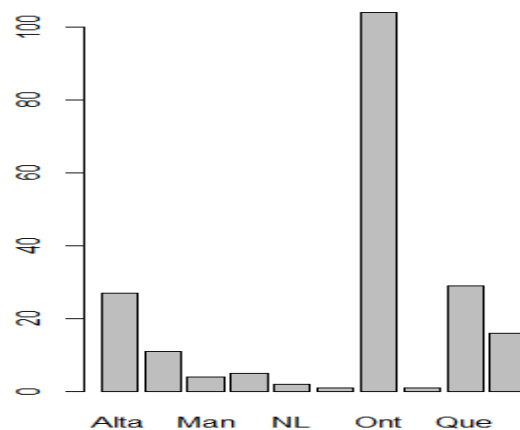
datos<- data.frame(datos)
height<- data.frame(datos$Height)
arm.span<- data.frame(datos$Arm.Span)
foot.size<- data.frame(datos$Foot.Size)
matriz<- c(height, arm.span, foot.size)

## matriz de correlacion
cor(height, arm.span)

##te devuelve el indice de las posiciones ordenadas
(ii<- order(height, arm.span, foot.size))
cbind(height, arm.span, foot.size)[ii,]
plot(datos$Province)

#Ejercicio 3. MapReduce. Simulación
#-----

#Con los datos de "student_census.csv" del ejercicio anterior, se pide:
#Particionar el dataset en 20 subconjuntos.
```



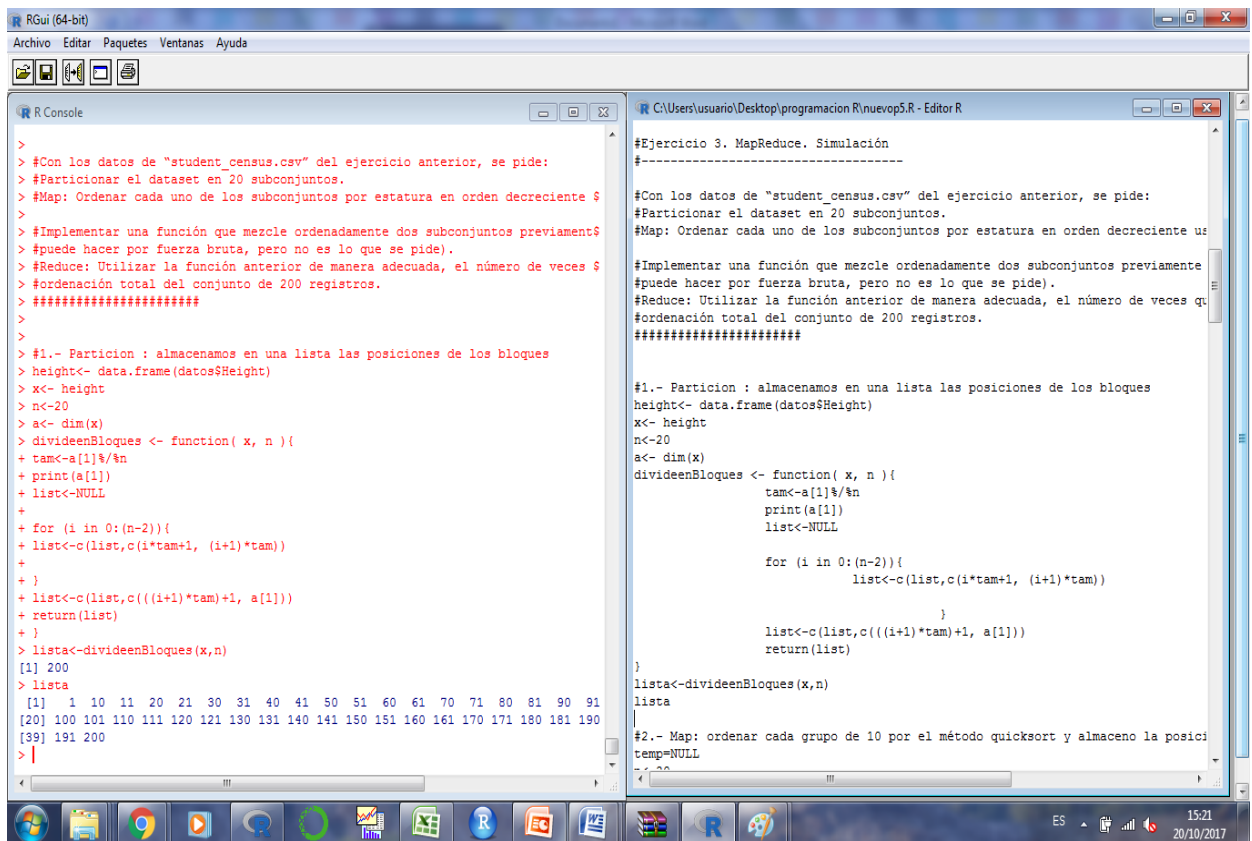
Ejercicio 3. MapReduce.

Simulación Con los datos de “student_census.csv” del ejercicio anterior, se pide: Particionar el dataset en 20 subconjuntos. Map: Ordenar cada uno de los subconjuntos por estatura en orden decreciente usando el algoritmo quicksort Implementar una función que mezcle ordenadamente dos subconjuntos previamente ordenados* (también se puede hacer por fuerza bruta, pero no es lo que se pide). Reduce: Utilizar la función anterior de manera adecuada, el número de veces que se requiera para conseguir la ordenación total del conjunto de 200 registros.

Solución

1. Paso 1: hago una función que devuelva una lista que indique donde empieza y donde acaba cada bloque:

- **Creo una lista vacía donde se almacenará el principio y el final de cada partición según el número de particiones deseadas n**



2.-MAP: Ordeno cada trozo del vector height de mayor a menor (orden descendiente) por el algoritmo QuickSort (nota:: Sé que existe una función para aplicar el algoritmo, pero yo lo he programado):

- Creo una función que tiene como parámetros el dataframe x (formado por la columna de alturas y otra columna con los índices para saber las posiciones del dataframe original cuando ordene x), la lista creada con la función anterior para saber las particiones y n, el número de subgrupos.
- Se realizan las operaciones en grupos de 10, en total 20 grupos (primer for)
- Se compara cada posición de la columna de alturas con el resto utilizando el algoritmo de ordenación quicksort, visto en los apuntes cómo funciona.
- Como resultado, se devuelve el vector x ordenado por alturas de 10 en 10 valores:

RGui (64-bit)

Archivo Editar Visualizar Misc Paquetes Ventanas Ayuda

R Console

```
+ }
+ list<-c(list,c(((i+1)*tam)+1, a[1]))
+ return(list)
+ }
> lista<-divideenBloques(x,n)
[1] 200
> lista
[1] 1 10 11 20 21 30 31 40 41 50 51 60 61 70 71 80 81 90 91
[20] 100 101 110 111 120 121 130 131 140 141 150 151 160 161 170 171 180 181 190
[39] 191 200
> #2.- Map: ordenar cada grupo de 10 por el método quicksort y almaceno la posi$
> temp=NULL
> n<-20
> x<- data.frame(height, list(1:200))
> pares_orden<-function (lista, x, n){
+   for (i in 0:(n-1)){
+     ini<-lista[i*2+1]
+     fin<-lista[i*2+2]
+     for (i in (ini:fin))
+     {
+       for (j in (ini:(fin-1)))
+       {
+         if (x[j, 1] < x[j+1, 1])
+         {
+           temp = x[j, ]
+           x[j, ]=x[j+1,]
+           x[j+1, ]=temp
+         }
+       }
+     }
+   }
+   return(x)
+ }
> a<- pares_orden(lista, x, n)
> |
```

R\Users\usuario\Desktop\programacion R\nuevop5.R - Editor R

```
list<-c(list,c(((i+1)*tam)+1, a[1]))
return(list)
}
lista<-divideenBloques(x,n)
lista
#2.- Map: ordenar cada grupo de 10 por el método quicksort y almaceno la posi
temp=NULL
n<-20
x<- data.frame(height, list(1:200))
pares_orden<-function (lista, x, n){
  for (i in 0:(n-1)){
    ini<-lista[i*2+1]
    fin<-lista[i*2+2]
    for (i in (ini:fin))
    {
      for (j in (ini:(fin-1)))
      {
        if (x[j, 1] < x[j+1, 1])
        {
          temp = x[j, ]
          x[j, ]=x[j+1,]
          x[j+1, ]=temp
        }
      }
    }
  }
  return(x)
}
a<- pares_orden(lista, x, n)

##Mezclar dos subconjuntos previamente ordenados::
conjunto1<- a[lista[1]:lista[2], 1]
```

ES 15:23 20/10/2017

RGui (64-bit)

Archivo Editar Visualizar Misc Paquetes Ventanas Ayuda

R Console

```
> a
datos.Height X1.200
1 184 9
2 178 7
3 175 6
4 173 4
5 168 1
6 167 8
7 163 2
8 163 3
9 158 5
10 150 10
11 177 17
12 176 11
13 176 15
14 173 19
15 170 12
16 167 13
17 166 16
18 166 20
19 162 14
20 155 18
21 211 24
22 181 23
23 181 25
24 180 26
25 173 28
26 173 30
27 166 22
28 164 27
29 162 21
30 153 29
31 185 39
32 170 32
33 170 33
34 170 35
```

R\Users\usuario\Desktop\programacion R\nuevop5.R - Editor R

```
list<-c(list,c(((i+1)*tam)+1, a[1]))
return(list)
}
lista<-divideenBloques(x,n)
lista
#2.- Map: ordenar cada grupo de 10 por el método quicksort y almaceno la posi
temp=NULL
n<-20
x<- data.frame(height, list(1:200))
pares_orden<-function (lista, x, n){
  for (i in 0:(n-1)){
    ini<-lista[i*2+1]
    fin<-lista[i*2+2]
    for (i in (ini:fin))
    {
      for (j in (ini:(fin-1)))
      {
        if (x[j, 1] < x[j+1, 1])
        {
          temp = x[j, ]
          x[j, ]=x[j+1,]
          x[j+1, ]=temp
        }
      }
    }
  }
  return(x)
}
a<- pares_orden(lista, x, n)

##Mezclar dos subconjuntos previamente ordenados::
conjunto1<- a[lista[1]:lista[2], 1]
```

ES 15:23 20/10/2017

Se puede observar como de la posición 1 a la 10, los números están ordenados de forma ascendente, de la posición 11 a la 20, ocurre lo mismo, y así con los 20 grupos.

Por otro lado, la 3 columna almacena las posiciones donde se encontraban en la matriz de datos.

3.- Ordenar 2 conjuntos:

Se crea una función donde dados dos grupos de 10, se cree un vector de longitud `length(grupo1)+length(grupo2)`, en este caso de 20, con sus componentes de altura ordenadas:

- Se inicializa un vector de longitud la suma de los 2 grupos
- Se va buscando cuál es la componente mayor de las alturas de los dos grupos y se almacena en el vector, ya que se quiere ordenar en forma descendente:

```
RGui (64-bit)
Archivo  Editor  Paquetes  Ventanas  Ayuda

R Console
+ contador<- 1
+ contador2<- 1
+ for (i in (1:(length(conjunto1))))
+ {
+ while (contador2<=(length(conjunto2)))
+ {
+ if (conjunto1[i] >= conjunto2[contador2])
+ {
+ vectord[contador]<- conjunto1[i]
+ contador=contador+1
+ break }
+ if (conjunto1[i] <= conjunto2[contador2])
+ {
+ vectord[contador]<- conjunto2[contador2]
+ contador = contador+1
+ contador2 = contador2+1 }
+ }
+ }
+ if (conjunto1[length(conjunto1)] > conjunto2[length(conjunto2)])
+ {
+ vectord[length(vectord)]<-conjunto2[length(conjunto2)]
+ }
+ else
+ {
+ vectord[length(vectord)]<-conjunto1[length(conjunto1)]
+ }
+ }
+ return(vectord)
+ }
+ }
> ordenar2conjuntos(conjunto1, conjunto2)
[1] 184 178 177 176 176 175 173 173 170 168 167 167 166 166 163 163 162 158 155
[20] 150
>
> |

C:\Users\usuario\Desktop\programacion R\nuevo5.R - Editor R
##Mezclar dos subconjuntos previamente ordenados::

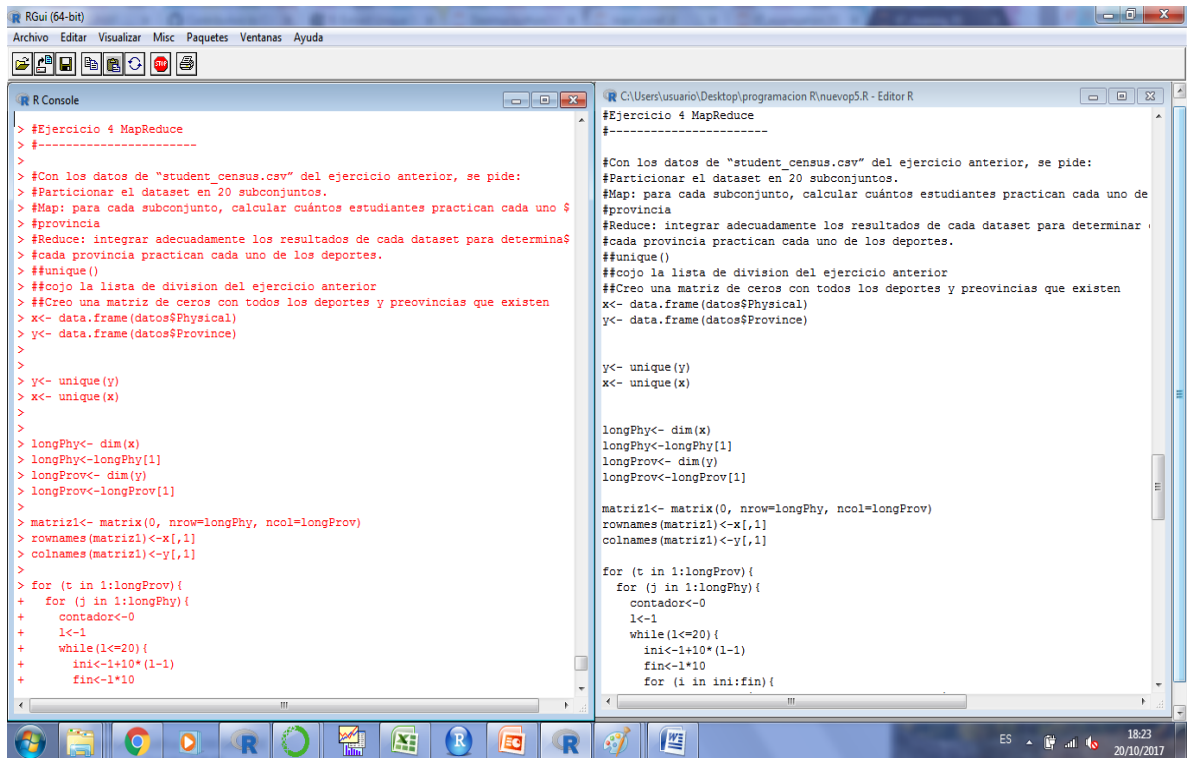
conjunto1<- a[lista[1]:lista[2], 1]
conjunto2<- a[lista[3]:lista[4], 1]
vectord<-NULL
ordenar2conjuntos<- function(conjunto1, conjunto2)
{
  vectord<- (1:(length(conjunto2)+length(conjunto1)))
  contador<- 1
  contador2<- 1
  for (i in (1:(length(conjunto1))))
  {
    while (contador2<=(length(conjunto2)))
    {
      if (conjunto1[i] >= conjunto2[contador2])
      {
        vectord[contador]<- conjunto1[i]
        contador=contador+1
        break }
      if (conjunto1[i] <= conjunto2[contador2])
      {
        vectord[contador]<- conjunto2[contador2]
        contador = contador+1
        contador2 = contador2+1 }
      }
    }
    if (conjunto1[length(conjunto1)] > conjunto2[length(conjunto2)])
    {
      vectord[length(vectord)]<-conjunto2[length(conjunto2)]
    }
    else
    {
      vectord[length(vectord)]<-conjunto1[length(conjunto1)]
    }
  }
  return(vectord)
}
```

En esta función tomo los dos primeros grupos (ya ordenados) de la matriz a, es decir, los 20 primeros números.

Ésta programada ordena dos grupos ya previamente ordenados y devuelve un vector que contiene los dos grupos ordenados.

4.- Por último, aplicamos el método REDUCE, para ello implementaré una función que llame a la función anterior "ordenar2grupos(conjunto1, conjunto2)" en donde en cada vuelta el conjunto1 será el ordenado en la vuelta anterior y el conjunto2 el conjunto nuevo que se añade en cada iteración:

Como se puede observar con el comando sum, el bucle ha recorrido las 200 observaciones y ha almacenado todas en la matriz creada, ya que la suma de todas las componentes de la matriz es 200.



```
> #Ejercicio 4 MapReduce
> #-----
>
> #Con los datos de "student_census.csv" del ejercicio anterior, se pide:
> #Particionar el dataset en 20 subconjuntos.
> #Map: para cada subconjunto, calcular cuántos estudiantes practican cada uno de
> #provincia
> #Reduce: integrar adecuadamente los resultados de cada dataset para determinar
> #cada provincia practican cada uno de los deportes.
> ##unique()
> ##Cojo la lista de division del ejercicio anterior
> ##Creo una matriz de zeros con todos los deportes y preovincias que existen
> x<- data.frame(datos$Physical)
> y<- data.frame(datos$Province)
>
>
> y<- unique(y)
> x<- unique(x)
>
>
> longPhy<- dim(x)
> longPhy<-longPhy[1]
> longProv<- dim(y)
> longProv<-longProv[1]
>
> matriz1<- matrix(0, nrow=longPhy, ncol=longProv)
> rownames(matriz1)<-x[,1]
> colnames(matriz1)<-y[,1]
>
> for (t in 1:longProv){
+   for (j in 1:longPhy){
+     contador<-0
+     l<-1
+     while(l<=20){
+       ini<-1+10*(l-1)
+       fin<-l*10
+     }
+   }
+ }
```

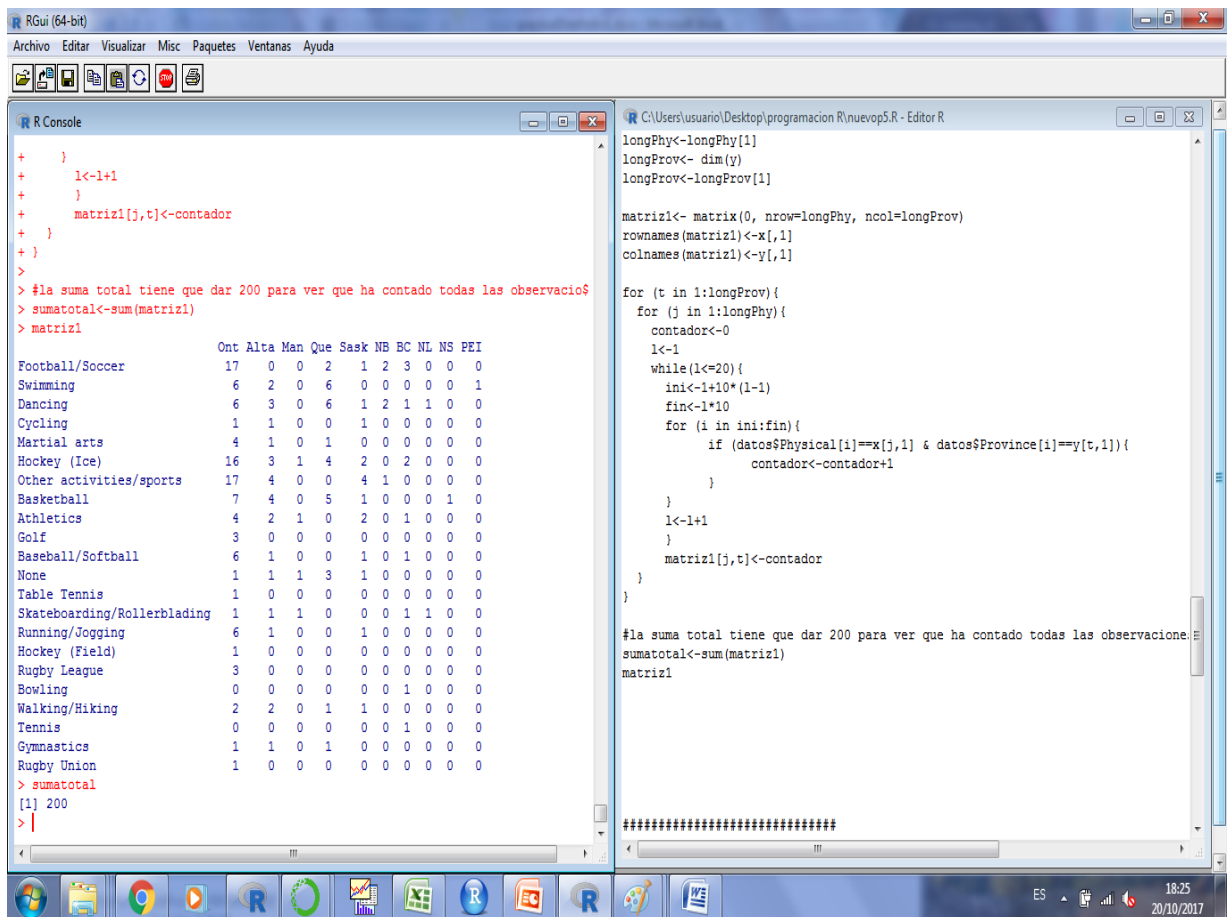
```
#Ejercicio 4 MapReduce
#-----
#Con los datos de "student_census.csv" del ejercicio anterior, se pide:
#Particionar el dataset en 20 subconjuntos.
#Map: para cada subconjunto, calcular cuántos estudiantes practican cada uno de
#provincia
#Reduce: integrar adecuadamente los resultados de cada dataset para determinar
#cada provincia practican cada uno de los deportes.
##unique()
##Cojo la lista de division del ejercicio anterior
##Creo una matriz de zeros con todos los deportes y preovincias que existen
x<- data.frame(datos$Physical)
y<- data.frame(datos$Province)

y<- unique(y)
x<- unique(x)

longPhy<- dim(x)
longPhy<-longPhy[1]
longProv<- dim(y)
longProv<-longProv[1]

matriz1<- matrix(0, nrow=longPhy, ncol=longProv)
rownames(matriz1)<-x[,1]
colnames(matriz1)<-y[,1]

for (t in 1:longProv){
  for (j in 1:longPhy){
    contador<-0
    l<-1
    while(l<=20){
      ini<-1+10*(l-1)
      fin<-l*10
      for (i in ini:fin){
```



Ejercicio 5. MapReduce.

Medias por bloques Implementar una función que reciba como entrada un dataset de números y un número natural. La función debe devolver la media de los números almacenados en el dataset. Para ello se debe implementar una simulación de la metodología MapReduce de la siguiente forma: # Primero se recogen los datos y el valor n del input # Segundo se divide el dataset en n datasets de tamaños aproximados

Solución:

Primero se crea una función (como la del ejercicio 3) donde se introduce el número de grupos que se quiere hacer y el vector. Ésta función devuelve una lista con las posiciones donde empieza y acaba cada uno de los grupos para aplicar el "map".

Map: A cada trozo, definido por la lista anterior, se suma el valor de cada dato y se almacena el número de componentes que hay. Se obtiene una tupla donde la primera componente es la suma de cada uno de los valores y la segunda componente es la longitud de cada trozo.

Reduce: Se suma la primera componente de cada una de éstas tuplas, se suma la segunda componente de cada una de éstas tuplas.

Por último se divide la primera componente (suma de los valores) entre la segunda componente (longitud total) y de ésta forma se obtiene la media.

```
RGui (64-bit)
Archivo  Editar  Visualizar  Misc.  Paquetes  Ventanas  Ayuda

R Console
> #Ejercicio 5.
> #-----
>
> x<- c(1:80, 78:34, 1,2,3,4,5,6,7,5,4,3,2)
>
> n<-5
>
>
> divideenBloques <- function( x, n ){
+ tam<-length(x)%/%n
+ list<-NULL
+
+ for ( i in 0:(n-2)){
+   list<-c(list,c(i*tam+1, (i+1)*tam))
+ }
+ list<-c(list,c(((i+1)*tam)+1, length(x)))
+ return(list)
+ }
> lista<-divideenBloques(x,n)
> #####
> i<-1
> pares_sumas<-function (lista, x, n){
+   sumas<-NULL
+   tams<-NULL
+   for ( i in 0:(n-1)){
+     ini<-lista[i*2+1]
+     fin<-lista[i*2+1]
+     sumas<-c(sumas, sum(x[ini:fin]))
+     tams<-c(tams, fin-ini+1)
+   }
+   return(data.frame(sumas, tams))
+ }
> ps<-pares_sumas(lista, x, n)
>
>
> reduce<-function(ps){
+
+ }
```

```
Archivo  Editar  Visualizar  Misc  Paquetes  Ventanas  Ayuda
[Icons]

R Console
+ tam<-length(x)%4*n
+ lista<-NULL
+
+ for (i in 0:(n-2)){
+   lista<-c(lista,c(i*tam+1, (i+1)*tam))
+ }
+ lista<-c(lista,c(((i+1)*tam)+1, length(x)))
+ return(lista)
+ }
> lista<-divideenBloques(x,n)
> #####
> i<-1
> pares_sumas<-function (lista, x, n){
+   sumas<-NULL
+   tams<-NULL
+   for (i in 0:(n-1)){
+     ini<-lista[i*2+1]
+     fin<-lista[i*2+2]
+     sumas<-c(sumas, sum(x[ini:fin]))
+     tams<-c(tams,fin-ini+1)
+   }
+   return(data.frame(sumas, tams))
+ }
> ps<-pares_sumas(lista, x, n)
>
> reduce<-function(ps){
+   resultado<- as.double(sum(ps$sumas)/sum(ps$tams))
+   return(resultado)
+ }
> resultado_media<-reduce(ps)
> resultado_media
[1] 42.66176
>

C:\Users\usuario\Desktop\programacion R\nuevoop5.R - Editor R
divideenBloques <- function(x, n){
  tam<-length(x)%4*n
  lista<-NULL

  for (i in 0:(n-2)){
    lista<-c(lista,c(i*tam+1, (i+1)*tam))
  }

  lista<-c(lista,c(((i+1)*tam)+1, length(x)))
  return(lista)
}

lista<-divideenBloques(x,n)
#####
i<-1
pares_sumas<-function (lista, x, n){
  sumas<-NULL
  tams<-NULL
  for (i in 0:(n-1)){
    ini<-lista[i*2+1]
    fin<-lista[i*2+2]
    sumas<-c(sumas, sum(x[ini:fin]))
    tams<-c(tams,fin-ini+1)
  }
  return(data.frame(sumas, tams))
}
ps<-pares_sumas(lista, x, n)

reduce<-function(ps){
  resultado<- as.double(sum(ps$sumas)/sum(ps$tams))
  return(resultado)
}
resultado_media<-reduce(ps)
```

Ejercicio 6. Programación Recursiva y Funcional

Observad este código Haskell: `> [(l,h)|h<-[0..51], l<-[0..h-1], sum[l..h]==100]` Intentar adivinar cuál es el propósito del código y qué resultado obtiene. Observar esta solución recursiva en R y decidir cuál es el propósito del código: `busca <- function(l, h, st){ if (l<=st) {if (sum(l:h) ==st) {print(c(l, h)) return(busca(l+1,l+1,st))} else {if (sum(l:h)`

Solución

Es una función recursiva, en el enunciado se nos da la condición de que l y h son dos números naturales donde h está entre 0 y 51 y l entre 0 y $h-1$. El conjunto que se define en el enunciado son los l, h tales que la suma de los números que se encuentran entre l y h (inclusive) suman 100.

La función que se nos pide ejecutar en R, busca los l, h tales que su suma==st (en este caso 100).

#Se introducen los parámetros l, h y el valor que se quiere que sumen los números entre l y h (st):

```
busca <- function (l, h, st){
```

#Siempre que l sea menor que la suma total(en este caso 100):

```
if (l<=st)
```

si la suma es st, entonces pinta la terna l, h y vuelve a llamar a la función

#con $l = l+1$ y $h = l+1$, para poder buscar otras tuplas que lo cumplan empezando a mirar la siguiente posición de l y poniendo $h=l+1$ para poderla ir incrementando en las siguientes vueltas

```
{if (sum(l:h) ==st) {print(c(l, h))  
    return(busca(l+1,l+1,st))}}
```

#Sino:

##En caso de que la suma de l y h sea menor que st, se sigue incrementando el valor de h , por lo que se vuelve a llamar a la función incrementando 1 a la h

En caso contrario, la suma de l y h es mayor que st por lo que no se ha encontrado para l una h que cumpla que $\text{sum}(l:h)=st$, en este caso se vuelve a llamar a la función con $(busca(l+1, l+1, st))$ para volver a buscar una tupla

```
else {if (sum(l:h)<st) return(busca(l, h+1, st))  
    else return(busca(l+1, l+1, st))}}
```

```
}
```

```
}
```

```
busca(0,100, 100) #Devuelve el conjunto de tuplas que cumplen  $\text{sum}(l:h)=st$ 
```

