Índex

Capítol 1

Repàs: Funcions, Vectors i Esquemes de Recorregut i Cerca

Objectius: Recordar i enfortir els coneixements adquirits a l'assignatura d'Introducció a la Informàtica, que seran bàsics per a poder continuar amb aquesta assignatura.

Primer de tot:

- Creeu una carpeta (o working directory) que es digui `Repas` on emmagatzemareu tots els scripts que codifiquen les diferents funcions d'aquesta part.
- Recordeu d'anar a aquest directori utilitzant `setwd("..../Repas")`
- Aconsellem quardar cada funció dins el directori `Repas` en un fitxer que es digui igual que la funció seguit d'un punt i `R`. Per exemple si la funcio es diu `mi_fu` enmmagatzemeu en un script `mi_fu.R`.

1.1 Exercicis

1. Fes una funció que donat un vector d'enters, retorni quin és el primer múltiple de 3 que hi ha a la seqüència, o 0 si no hi ha cap.

```
1  triple ← function (v)
2  {
3      i ← 1
4      n ← length(v)
5      while ((i<=n) && ((v[i]%%3)!=0)) {
6            i ← i+1
7      }
8      if (i>n) { return (0) }
9      else { return (v[i]) }
10 }
11
```

```
# cat (triple(c(5,7,4,13,12,4,3,6,2,1)),"\n")
# cat (triple(c(5,4,4,7,8,10,11)),"\n")
```

2. Dissenya una funció que donat un vector d'enters retorni un altre vector d'enters amb els elements del primer en ordre invers.

3. Dissenya una funció que donats dos vectors d'enters v1 i v2 ordenats, retorni un tercer vector que conté tots els elements de v1 i v2 també ordenats (si hi ha repetits es repeteixen també en el vector resultat).

```
barreja ← function (v1, v2)
{
    r ← vector()
    n1 ← length(v1)
    n2 ← length(v2)
    i ← 1
    j ← 1
    while ((i <= n1) && (j <= n2)) {
          if (v1[i] <= v2[j]) {
                r ← c(r,v1[i])
           i ← i+1
          }
          else {
                r ← c(r,v2[j])
           j ← j+1
          }
    }
    if (i > n1) {
          for (k in j:n2) {
                r ← c(r, v2[k])
          }
    }
    else {  # j > n2
          for (k in i:n1) {
                r ← c(r, v1[k])
          }
    }
    return  (r)
}

# v1 ← c(1,1,4,4,5,6,9)
# v2 ← c(1,2,3,5,7,9,10)
# cat (barreja(v1,v2),"\n")

# v1 ← c(1,1,4,4,5,6,9)
# v2 ← c(1,2,3,5,7,9,10,12,12,14,15)
# cat (barreja(v1,v2),"\n")

# v1 ← c(1,1,4,4,5,6,9,11,13,14,17)
# v2 ← c(1,2,3,5,7,9,10)
# cat (barreja(v1,v2),"\n")
```

4. Fes una funció que donat un vector d'enters i un enter, retorni TRUE si i només si l'enter donat és divisor d'algun dels elements del vector.

```
esdivisor ← function (v, e)
{
    n ← length(v)
    i ← 1
    while ((i <= n) && ((v[i]%%e) != 0)) {
            i ← i + 1
    }
    return (i <= n)
}

v ← c(1,5,3,18,4,11,21)

# cat (esdivisor(v,3),"\n")
# cat (esdivisor(v,2),"\n")
# cat (esdivisor(v,8),"\n")
# cat (esdivisor(v,11),"\n")
```

5. Fes una funció que donades dues cadenes de caràcters que representen dues paraules o frases diferents, retorni TRUE si aquestes dues paraules són bifronts o falsos palíndroms, és a dir que es llegeixen igual una del dret i l'altra del revés. Per exemple: roma - amor.

6. Fes una funció que donades dues cadenes de caràcters que representen dues paraules, retorni TRUE si aquestes dues paraules són anagrames. Una paraula és anagrama d'una altra si i només si, els caràcters que tenen totes dues paraules són els mateixos, encara que estiguin ordenats de forma diferent. Per exemple: roma - amor - omar - mora, són anagrames totes 4 entre sí, i arbol - labor - borla, també ho són.

```
anagrames ← function (a, b)
{
    # convertim primer les paraules en vectors de
       caracters
    v1 ← substring (a, 1:nchar(a), 1:nchar(a))
    v2 ← substring (b, 1:nchar(b), 1:nchar(b))
    n ← length(v1)
    if (n != length(v2)) {
          return (FALSE)
    }
    else { # ordenem caracters i comparem un a un
          v1sort ← sort(v1)
          v2sort ← sort(v2)
          i ← 1
          while ((i<=n)&&(v1sort[i]==v2sort[i])) {
                  i ← i + 1
          }
          return (i > n)
    }
```

```
}

# a <- "roma"
# b <- "amor"
# cat (anagrames(a,b),"\n")

# a <- "arbol"
# b <- "labor"
# cat (anagrames(a,b),"\n")

# a <- "arbre"
# b <- "rebra"
# cat (anagrames(a,b),"\n")

# a <- "mobil"
# b <- "rebre"
# cat (anagrames(a,b),"\n")
```

7. Tenim la informació dels primers i segons classificats al Campionat Nacional de Lliga de futbol des dels seus inicis (any 1928) en diversos retalls de paper. Però aquesta informació, encara que cronològicament és correcta: sabem qui va ser el primer campió de la primera lliga, i després apareix en algun moment el campió de la 2a lliga abans del campió de la 3a lliga, però estan barrejats amb els subcampions.

   Així doncs l'ordre de les lligues es manté. Primer apareix la informació del campió o subcampió d'una lliga abans de les dades de la següent lliga, però estan barrejats els campions i subcampions.

   Feu un programa que, donada una seqüència de les dades de les lligues, en format de nom de l'equip i posició (1 o 2), que acabarà en un equip fictici "FI", s'ha de generar la taula dels campions i subcampions de lliga de forma cronològica però aparellant correctament els campions i els subcampions del mateix any. La informació és correcta i hi ha el mateix nombre de campions i de subcampions.

   Per exemple, si l'entrada fos:

```
Barça 1
Ath-Bilbao 2
Madrid 2
At-Madrid 2
València 1
Espanyol 1
At-Madrid 1
Barça 2
FI
```

El resultat seria:

```
1er i 2on CLASSIFICATS DE LES LLIGUES
-------------------------------------
Campions          Subcampions
--------          -----------
Barça             Ath-Bilbao
València          Madrid
Espanyol          At-Madrid
At-Madrid         Barça
```

8. Escriu una acció que, donat un vector d'enters, escriu per pantalla, en ordre invers a com arriben, tots els enters que siguin senars havent-los multiplicat abans pel senar mínim dels nombres del vector. Per exemple:

   Si el vector és: `3 5 4 8 7 9 10 12 5 2 6 31 22 3 9 0`

   El resultat a la sortida ha de ser: `27 9 93 15 27 21 15 9` *(ja que el mínim senar és el 3)*

```
1  senarsinvers ← function (v)
2  {
3      n ← length(v)
4
5      # primer hem de trobar el senar minim
6      i ← 1
7      while ((i <= n) && ((v[i]%%2) == 0)) {
8            i ← i + 1
9      }
10     if (i > n) {
11           # no hi ha cap senar al vector
12           cat ("El vector no ét cap senar\n")
13     }
14     else {
15           min ← v[i]
16           for (j in i:n) {
17                 if ((v[j] < min) && ((v[j]%%2) !=
                       0)) {
18                   min ← v[j]
19                 }
20           }
21
22           # i ara escrivim la sortida per pantalla
23           for (j in n:i) {
24                 if ((v[j]%%2) != 0) {
25                       cat (v[j]*min, "")
26                 }
27           }
28           cat ("\n")
29         }
30 }
31
```

```
# v <- c(3,5,4,8,7,9,10,12,5,2,6,31,22,3,9,0)
# senarsinvers(v)

# v <- c(4,8,10,12,2,6,22,1)
# senarsinvers(v)

# v <- c(4,8,10,12,2,6,22,10)
# senarsinvers(v)
```

9. *Quicksort*. (Un clàssic). La funcio `qs(v)` ha d'ordenar una taula de nombres, per exemple

```
> v<-c(1,5,2,3,4,7,10,9, 6)
> v
[1]  1  5  2  3  4  7 10  9  6
> qs(v)
[1]  1  2  3  4  5  6  7  9 10
```

Us demanem que completeu el codi:

```
qs <- function(vec){
  if(length(vec) > 1){
    pivot <- vec[1]
    low <- qs(vec[vec < pivot])
    mid <- vec[vec == pivot]
    high <- ....)
    return (c(low, ..., ....))
  }
  else { return (vec) }
}
```

Recordeu que cal emmagatzemar la funció dins `Repas` en un script (que recomanem) es digui `qs.R`

```
#Una possible solucio
qs <- function(vec){
  if(length(vec) > 1){
    pivot <- vec[1]
    low <- qs(vec[vec < pivot])
    mid <- vec[vec == pivot]
    high <- qs(vec[vec > pivot])
    return (c(low, mid, high))
  }
  else { return (vec) }
}
```

Capítol 2

Matrius

Objectius: Aprendre a treballar amb l'estructura de dades `matrix`. Per això es dissenyaran un conjunt de funcions que permetin resoldre problemes bàsics sobre aquesta estructura de dades. S'utilitzaran algunes d'aquestes funcions per introduir (o recordar) algunes construccions bàsiques de l'estadística. Finalment es consideraran alguns casos de mostreig.

Primer de tot:

- Creeu una carpeta (o working directory) que es digui `Matrius` i emmagatzameu dins tots els fitxers que codifiquen les diferents funcions d'aquesta part.
- Recordeu d'anar a aquest directori utilitzant `setwd("..../Matrius")`

2.1 Exercicis

1. *QuantsCops*. Feu una funció que donat un enter i una matriu retorni el nombre de vegades que apareix l'enter en la matriu.

```
quantesvegades ← function (A,n) {
   nvegades ← 0
   for (i in 1:nrow(A)) {
      for (j in 1:ncol(A)) {
          if (A[i,j] == n) {
            nvegades ← nvegades+1
             }
      }
   }
  return(nvegades)
}

#col ← scan(n=1)  #nombre de columnes de la matriu
#v ← scan()     #valors de la matriu
#n ← scan(n=1) #enter que volem buscar
#A ← matrix(v,ncol=col) #construim la matriu

```

```
18 cat ("El nombre de vegades que el nombre", n, "
       apareix en la matriu es",quantesvegades(A,n) ,"
       \n")
```

2. *Duplicats*. Feu una funció que rebi una matriu i la retorni amb tots els seus elements duplicats.

```
1 duplica ← function(A){
2    for (i in 1:nrow(A)) {
3       for (j in 1:ncol(A)) {
4          A[i,j] ← 2*A[i,j]
5       }
6    }
7    return(A)
8 }
```

3. *GeneraDiagonal*. Feu una funció que rebi un vector de dimensió n i retorni una matriu de n x n que tingui aquest vector a la diagonal i a tota la resta zeros.

```
1 vect_diag ← function(v){
2    c ← rep(0,length(v)*length(v)) #construim el
        vector de zeros
3    A ← matrix(c,ncol=length(v))  #construim la
        matriu de zeros
4    for (i in 1:nrow(A)) {
5          A[i,i] ← v[i]
6    }
7    return(A)
8 }
```

4. *ElementsDiagonal*. Feu una funció que rebi una matriu quadrada i retorni un vector amb els elements de la diagonal de la matriu.

5. *CaVertical*. Aquest exercici es compon de vàries funcions a dissenyar:

   (a) Dissenyeu una funció `mat_lletres` que donats dos nombres, corresponents al nombre de columnes i de files de la matriu, ompli aleatòriament de lletres $\{\texttt{a}, \texttt{b}, \texttt{c}\}$ una matriu i la retorni. Aconsellem que l'script on escriviu la funció es digui igual que la funció.

   Com a exemple d'execució tindríeu:

```
>  setwd ("..../Matrius")
>  source ("mat_lletres.R")
>  M <- mat_lletres(5,10)
> M
      [,1] [,2] [,3] [,4] [,5]
 [1,] "b"  "c"  "c"  "c"  "a"
 [2,] "b"  "a"  "b"  "b"  "a"
 [3,] "c"  "c"  "b"  "a"  "b"
 [4,] "b"  "c"  "c"  "b"  "b"
```

```
 [5,] "b"  "a"  "b"  "a"  "b"
 [6,] "b"  "b"  "c"  "b"  "c"
 [7,] "a"  "a"  "a"  "a"  "b"
 [8,] "a"  "c"  "c"  "b"  "c"
 [9,] "b"  "a"  "c"  "a"  "c"
[10,] "b"  "a"  "c"  "b"  "a"
```

(b) Dissenyeu una funció `quantes_ca` que donada una matriu de lletres calcula quantes vegades apareix (en vertical) la combinació "ca". La matriu haurà estat generada aplicant `mat_lletres`.

Com a exemple d'execució tindríeu:

```
> M
      [,1] [,2] [,3] [,4] [,5]
 [1,] "b"  "c"  "c"  "c"  "a"
 [2,] "b"  "a"  "b"  "b"  "a"
 [3,] "c"  "c"  "b"  "a"  "b"
 [4,] "b"  "c"  "c"  "b"  "b"
 [5,] "b"  "a"  "b"  "a"  "b"
 [6,] "b"  "b"  "c"  "b"  "c"
 [7,] "a"  "a"  "a"  "a"  "b"
 [8,] "a"  "c"  "c"  "b"  "c"
 [9,] "b"  "a"  "c"  "a"  "c"
[10,] "b"  "a"  "c"  "b"  "a"
> source("quantes_ca.R")
>  n<-quantes_ca(M)
> n
[1] 5
```

(c) Donat que executant varies vegades `mat_lletres` s'obtenen resultats diferents (la generació de la matriu és aleatòria) com es pot veure en l'exemple següent:

```
> M1<-mat_lletres(5,10)
> M2<-mat_lletres(5,10)
> M1
      [,1] [,2] [,3] [,4] [,5]
 [1,] "b"  "b"  "a"  "c"  "c"
 [2,] "b"  "b"  "a"  "c"  "a"
 [3,] "a"  "c"  "c"  "c"  "a"
 [4,] "a"  "c"  "c"  "b"  "b"
 [5,] "a"  "c"  "b"  "a"  "b"
 [6,] "c"  "c"  "b"  "a"  "a"
 [7,] "a"  "b"  "c"  "b"  "a"
 [8,] "c"  "b"  "b"  "a"  "a"
 [9,] "a"  "c"  "a"  "b"  "a"
[10,] "a"  "c"  "b"  "a"  "a"
> M2
      [,1] [,2] [,3] [,4] [,5]
 [1,] "b"  "c"  "a"  "c"  "c"
 [2,] "c"  "a"  "a"  "c"  "b"
```

```
 [3,] "a"  "b"  "b"  "b"  "c"
 [4,] "c"  "c"  "b"  "a"  "a"
 [5,] "b"  "b"  "a"  "a"  "c"
 [6,] "c"  "c"  "b"  "c"  "c"
 [7,] "a"  "a"  "a"  "b"  "b"
 [8,] "c"  "b"  "a"  "c"  "b"
 [9,] "a"  "a"  "c"  "a"  "b"
[10,] "a"  "c"  "a"  "b"  "a"
>
```

Dissenyeu una funció `mostra_ca(s,c,f)` que a partir del nombre de columnes `c` i de files `f` i d'un nombre `s` que indica el nombre de resultats que es volen, retorni un vector de `s` components on cada component `i` s'obté amb el resultat d'aplicar la funció `quantes_ca` a una nova matriu generada amb `mat_lletres(c,f)`. És a dir, per a calcular cada component del vector resultant s'ha de generar una nova matriu amb `mat_lletres(c,f)` i executar `quantes_ca` sobre aquesta matriu.

6. *Combinacio31H.* Feu una funció que rebi una matriu amb 1, 2 i 3 i retorni quantes vegades hi apareix (en horitzontal) la combinació 31.

```
1  trentau ← function(A){
2     sum ← 0
3     for (i in 1:nrow(A)) {
4        for (j in 1:(ncol(A)-1)) {
5           if (A[i,j]==3 && A[i,j+1]==1) {
6              sum ← sum+1
7           }
8        }
9     }
10    return(sum)
11 }
12
13 #cat("La combinacio 31 apareix", trentau(A), "
       vegades \n")
```

7. *Notes.* Disposem d'una matriu amb les notes de tots els estudiants en les assignatures del curs (matriu estudiants/assignatures amb contingut numèric que és la nota).

   Es demanen tres funcions que donada la matriu de notes de tots els estudiants calculin respectivament:

   a) La mitjana de notes per a cada estudiant (retorna un vector amb totes les mitjanes).

   b) La mitjana de notes per a cada assignatura (retorna un vector amb totes les mitjanes).

   c) La mitjana de notes de tots els estudiants, és a dir la mitjana dels resultats de a) (retorna un valor).

Nota: No es pot usar la funció `mean` *de R*

8. *Matriu dispersa.* (Examen parcial 13/14). Direm que una matriu és **dispersa** quan la quantitat de zeros que té la matriu supera el 70% dels nombres totals de la matriu (el 70% es calcula multiplicant el nombre total d'elements de la matriu per 0.7).

   Donada aquesta descripció, fes una funció que donada una matriu retorni TRUE si aquesta matriu és dispersa i FALSE en cas contrari.

```
dispersa ← function (mat)
{
    nfiles ← nrow(mat)
    ncols ← ncol(mat)
    nzeros ← 0
    fila ← 1
    trobat ← FALSE
    while (fila <= nfiles && !trobat) {
        col ← 1
        while (col <= ncols && !trobat) {
            if (mat[fila,col]==0) {
                nzeros ← nzeros + 1
                if (nzeros > (nfiles*ncols*0.7)) {
                    trobat ← TRUE
                }
            }
            col ← col + 1
        }
        fila ← fila + 1
    }
    return (trobat)
}
```

9. *Combinacio23.* Feu una funció que rebi una matriu amb 1, 2 i 3 i retorni quantes vegades hi apareix la combinació 23 (en qualsevol direcció: horitzontal, vertical o diagonal).

```
vintitres ← function(A){
   sum ← 0
   for (i in 1:(nrow(A)-1)) {
      for (j in 1:(ncol(A)-1)) {
         if (A[i,j]==2) {
            if (A[i,j+1]==3) {       #horitzontal
               sum ← sum+1
            } else if (A[i+1,j]==3) {  #vertical
               sum ← sum+1
            } else if(A[i+1,j+1]==3) { #diagonal
               sum ← sum+1
            }
         }
      }
```

```
15      }
16      #ultima fila
17      for (z in 1:(ncol(A)-1)) {
18          if (A[i+1,z]==2 && A[i+1,z+1]==3) {
19              sum ← sum+1
20          }
21      }
22      #ultima columna
23      for (t in 1:(nrow(A)-1)) {
24          if (A[t,j+1]==2 && A[t+1,j+1]==3) {
25              sum ← sum+1
26          }
27      }
28      return(sum)
29  }
30
31  #cat("La combinacio 23 apareix", vintitres(A), "
        vegades \n")
```

10. *MaximaMitjana.* Feu una funció que rebi una matriu i retorni en un vector dos valors, el primer és l'index de la columna que té la mitjana més gran, i el segon és el valor d'aquesta mitjana.

11. *Transposada de Matrius.* Feu una funció que rep una matriu i en torna la seva transposada. Òbviament, no podeu fer servir la funció `t(M)`, però la podeu fer servir per verificar el vostre resultat.

12. *ColumnaOrdenada.* Dissenyeu una funció que donada una matriu A retorni `TRUE` si aquesta matriu és *columna-ordenada*. Una matriu és *columna-ordenada* si per a cada columna els seus elements estan ordenats de més petit a més gran, recorrent la columna per a cada fila. És a dir: $\forall i, j,\ A_{i,j} \leq A_{i+1,j}$.

13. *Matriu Simètrica.* Feu una funció que rep una matriu A quadrada i torna `TRUE` si i només si és una matriu simètrica. Una matriu és simètrica si i només si $\forall i, j\ A_{i,j} = A_{j,i}$.

```
1   simetrica ← function (M)
2   {
3       fil ← nrow(M)
4       col ← ncol(M)
5       # fem una cerca d'un valor diferent i
            recorrent nomes la meitat de la matriu
6       i ← 2
7       trobat ← FALSE
8       while (i <= fil && !trobat) {
9           j ← 1
10          while (j < i && !trobat) {
11              if (M[i,j] != M[j,i]) {
12                  trobat ← TRUE
13              }
```

```
14              j ← j+1
15          }
16          i ← i+1
17      }
18      return (!trobat)
19  }
20
21  M ← matrix(c(1,2,3,2,1,2,3,2,1), ncol=3)
22  cat (simetrica(M),"\n")
23
24  M ← matriu_s (3,3)
25  cat (simetrica(M),"\n")
26  cat (M,"\n")
```

14. *ModificaColumna.* Feu una funció que rebi una matriu i la retorni amb la seva última columna cambiada per la suma de les dues primeres columnes.

```
1  u_mes_dos ← function(A){
2     for (i in 1:nrow(A)) {
3        A[i,ncol(A)] ← A[i,1]+A[i,2]
4     }
5     return(A)
6  }
```

15. *Sobresurten.* Feu una funció que rebi una matriu i retorni quants dels seus elements són més grans que la mitjana de tots els elements de la matriu.

16. *Quadrat semi-màgic.* (Examen final 13/14) Fes una funció que donada una matriu quadrada de nombres enters més grans que 0, retorni cert si i només si cada una de les files de la matriu sumen 27 (és a dir, la suma dels elements per a cada fila és igual a 27).

    Nota: Aquest exercici té un punt extra si s'assoleix la màxima eficiència

```
1  semimagic ← function (mat) {
2      nfiles ← nrow (mat)
3      trobat ← FALSE
4      i ← 1
5      while (i <= nfiles && !trobat) {
6          j ← 1
7          sumfila ← 0
8          while (j <= nfiles && !trobat) {
9              sumfila ← sumfila + mat[i,j]
10             if (sumfila > 27) { trobat ← TRUE }
11             j ← j + 1
12         }
13         if (sumfila < 27) { trobat ← TRUE }
14         i ← i + 1
15     }
16     return (!trobat)
17 }
```

17. *MultMatrius.* Feu una funció que rebi dues matrius i en torni la seva multiplicació. Suposeu que les dimensions de les matrius són tals que es poden multiplicar. Podeu fer servir la instrucció de R A%*%B per comprovar que la vostra funció dóna el resultat correcte.

18. *TipusMatriu.* Dissenya una acció que, donada una matriu A, indiqui per pantalla si aquesta matriu s'ajusta a algun dels casos següents:

    a) La matriu és *Triangular superior*: Una matriu és triangular superior si: $A_{i,j} = 0$ per $j = 1, \ldots, n-1$ i $i = j+1, \ldots, n$.
    b) La matriu és *Diagonal*: Una matriu és diagonal si: $\forall i, j \ A_{i,j} = 0$ sempre que $i \neq j$.
    c) La matriu és *Triangular inferior*: Una matriu és triangular inferior si: $A_{i,j} = 0$ per $i = 1, \ldots, n-1$ i $j = i+1, \ldots, n$.

19. *Diagonal dominant.* (Examen parcial 14/15). Direm que una matriu és **diagonal dominant** quan per a cada fila el valor de la diagonal és més gran o igual que tots els altres elements de la mateixa fila.

    Donada aquesta descripció, fes una funció que donada una matriu retorni TRUE si aquesta matriu és diagonal dominant i FALSE en cas contrari.

```
1  diagdom ← function (mat)
2  {
3      nfiles ← nrow(mat)
4      ncols ← ncol(mat)
5      fila ← 1
6      trobat ← FALSE
7      while (fila <= nfiles && !trobat) {
8          diag = mat[fila,fila]
9          col ← 1
10         while (col <= ncols && !trobat) {
11             if (fila != col && mat[fila,col] >
                   diag) {
12                 trobat ← TRUE
13             }
14             col ← col + 1
15         }
16         fila ← fila + 1
17     }
18     return (!trobat)
19 }
```

20. *Diagonal ben ponderada.* (Examen final 14/15). Donada una matriu M quadrada de dimensió $n \times n$ amb $n \geq 3$ i un element (i, j) a l'interior (que no està a la vora) direm que $M[i, j]$ és *ben ponderat* si

    $M[i, j] \geq M[i-1, j] + M[i+1, j] + M[i, j-1] + M[i, j+1]$.

    Direm que M és *ben ponderada* si existeixen n punts interiors ben ponderats. Feu una funció que, donada la matriu i el nombre n, retorni TRUE si la matriu està ben ponderada i FALSE en cas contrari.

```
ponderada ← function (mat, n)
{
  nfiles ← nrow(mat)
  ncols ← ncol(mat)
  compt ← 0
  f ← 2
  trobat ← FALSE
  while (f <= nfiles-1 && !trobat) {
    c ← 2
    while (c <= ncols-1 && !trobat) {
      sum ← mat[f-1,c]+mat[f+1,c]+mat[f,c-1]+mat[f
          ,c+1]
      if (mat[f,c] >= sum) {
        compt ← compt + 1
        if (compt >= n) {
          trobat ← TRUE
        }
      }
      c ← c + 1
    }
    f ← f + 1
  }
  return (trobat)
}
```

21. *Què cerquem?.* (Examen final 14/15 - única). Tenim la següent funció de matrius ja implementada:

```
ex4 <- function(mat) {
    a = nrow(mat)
    b = ncol(mat)
    t <- c(rep(FALSE,b))
    for (j in 1:b) {
        i <- 1
        while (i <= a && !t[j]) {
            if (mat[i,j] < 0) {
                t[j] <- TRUE
            }
            i <- i+1
        }
    }
    return (t)
}
```

Si la matriu d'entrada d'aquesta funció és la següent:

```
 10  2  0  3 -5 22  3  4
  4 20  6 -7  0  5  2  2
 -9 23  7 33 77 13 15  5
 10 13 45  0 85  0 77 64
 12 15  9 -4 32 34 90 -2
```

quina serà la sortida de la funció?

La sortida de la funció serà un vector amb els valors següents:

```
1       TRUE FALSE FALSE TRUE TRUE FALSE FALSE TRUE
```

22. *Rotació de Matrius.* Feu una funció que rep una matriu (no necessàriament quadrada) i torna la `rotació` de la matriu. La rotació de la matriu és una altra matriu amb els mateixos valors però amb la diferència que se li ha aplicat una rotació en el sentit invers de les agulles del rellotge. Per exemple, si tenim la matriu:

```
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
```

la seva rotació (a l'esquerra) serà:

```
     [,1] [,2] [,3]
[1,]    2    4    6
[2,]    1    3    5
```

```
1  rotacio ← function (M)
2  {
3     MM ← matrix (rep(0,ncol(M)*nrow(M)),
4                  nrow=ncol(M),ncol=nrow(M))
5
6     for (i in 1:nrow(MM))
7     {
8        for (j in 1:ncol(MM))
9        {
10          MM[i,j] ← M[j,ncol(M)-i+1]
11       }
12    }
13    return (MM)
14 }
```

23. *DiaDia.* Feu una funció que donats dos vectors de la mateixa mida retorni una matriu quadrada que tingui aquests dos vectors com a diagonals superiors i inferiors i a la resta de posicions, 1.

```
1  diagonals ← function(v1,v2) {
2     col ← fil ← length(v1) + 1
3     N ← col - 1
4
5     A ← matrix(rep(1,col*fil),ncol=col)
6
7     for (i in 1:N) {
8        A[i,i+1] ← v1[i]
```

```
9         A[i+1,i] ← v2[i]
10    }
11
12    return(A)
13 }
14
15 d_sup ← sample(1:25,5)
16 d_inf ← sample(1:23,5)
17 diagonals(d_sup,d_inf)
```

24. *Matrius Superposades.* Feu una funció que rebi 2 matrius (no necessàriament de la mateixa mida) i que en torna la superposició.

    La superposició de dues matrius es defineix com la matriu que obtenim posant una matriu a sobre de l'altra (coincidint en la coordenada $(1,1)$) tal que en la posició (i,j) hi ha el màxim dels elements que hi ha en la mateixa posició en totes dues matrius. En cas que una matriu no tingui cap element en aquesta posició, assumirem que hi ha un zero.

    Per exemple, donades les matrius:

```
     [,1] [,2] [,3]
[1,]    2    4    6
[2,]    1    3    5
```

```
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
```

    la superposició és:

```
     [,1] [,2] [,3]
[1,]    2    4    6
[2,]    3    4    5
[3,]    5    6    0
```

25. *Sudoku.*

    Feu una funció que rep una matriu 9×9 amb nombres naturals del 1 al 9 i torna cert si i només si és un `sudoku`.

    Direm que una matriu és un sudoku si i només si per a cada fila (i per a cada columna) hi ha tots els nombres del 1 al 9 (òbviament, sense repetir). De fet, les regles del `sudoku` també demanen que per a cada quadrant 3×3 també hi ha tots els 9 nombres, però aquesta regla ara no la tindrem en compte per a aquest exercici.

```
repetits ← function (l) {

   l2 ← sort(l)

   for (i in 2:length(l2)) {
      if (l2[i-1] == l2[i]) {return (TRUE)}
   }
   return (FALSE)
}

sudoku ← function (m) {

   trobat ← FALSE
   i ← 1
   while (i <= 9 && !trobat) {
      trobat ← repetits (m[i,]) || repetits (m[,i
         ])
      i ← i + 1
   }

   return (!trobat)
}

# òaix és un sudoku:
f ← c(1,2,3,7,8,9,4,5,6,4,5,6,1,2,3,7,8,9,7,8,9,4,
5,6,1,2,3,2,3,1,8,9,7,5,6,4,5,6,4,2,3,1,8,9,7,8,9,
7,5,6,4,2,3,1,3,1,2,9,7,8,6,4,5,6,4,5,3,1,2,9,7,8,
9,7,8,6,4,5,3,1,2)

M ← matrix(f,ncol=9)
cat (sudoku(M),"\n")

M ← matriu_s()
cat (sudoku(M),"\n")
```

26. *Sudoku (II).*

    Feu una funció que rep una matriu 9×9 amb nombres naturals del 1 al 9 i torna cert si i només si és un `sudoku`.

    En aquest exercici, només verificarem la condició que demana que, per a cada submatriu 3×3 també hi hagi tots els 9 nombres.

```
sense_repetits ← function (M) {
# sabem que M es una matriu 3x3 que nomes pot
    tenir valors de 1 a 9.
# Per fer-la mes generica, caldria substituir el 3
     per ncol/nrow.

   v ← rep(FALSE,9)
   for (i in 1:3) {
      for (j in 1:3) {
```

```
8            if (v[M[i,j]]) {return (FALSE)}
9            else {v[M[i,j]] ← TRUE}
10        }
11     }
12     return (TRUE)
13  }
14
15  sudoku2 ← function (M) {
16     posicions ← c(1,4,7)
17     for (i in posicions) {
18        for (j in posicions) {
19           if (!sense_repetits (M[i:(i+2),j:(j+2)]))
20                                  { return (FALSE) }
21        }
22     }
23     return (TRUE)
24  }
25
26  # aixo es un sudoku:
27  f ← c(1,2,3,7,8,9,4,5,6,4,5,6,1,2,3,7,8,9,7,8,9,
28  4,5,6,1,2,3,2,3,1,8,9,7,5,6,4,5,6,4,2,3,1,8,9,7,
29  8,9,7,5,6,4,2,3,1,3,1,2,9,7,8,6,4,5,6,4,5,3,1,2,
30  9,7,8,9,7,8,6,4,5,3,1,2)
31
32  M ← matrix(f,ncol=9)
33  cat (sudoku(M),"\n")
34
35  M ← matriu_s()
36  cat (sudoku(M),"\n")
```

27. *VariacionsCa.* En aquest execici farem una "variació" de l'exercici *CaVertical*:

    (a) Dissenyeu una funció `quantes_ca_dia` que donada una matriu com la de l'exercici *CaVertical* calcula quantes vegades apareix la combinació"ca" en diagonal.

    (b) Ara dissenyeu una funció per generar mostres a partir de `quantes_ca_dia`. La diferencia, però, és que volem passar la funció com a paràmetre. Cal que completeu (substituint els punts suspensius pel que calgui) el codi de `mostra_general` (en què `F` és una funció qualsevol que rep una matriu i retorna un enter):

```
mostra_general<-function(s,c,f, F){
   v<-1:s
   for(i in 1:s){
      M<-mat_lletres(c,f)
      v[i]<- .....
   }
   return(v)
}
```

Mireu que tot funcioni com cal, executant per exemple:

```
> source("quantes_ca.R")
> m1<-mostra_general(5, 10, 10, quantes_ca)
> m1
[1] 14 10 10 10 16
> source("quantes_ca_dia.R")
> m2<-mostra_general(5, 10, 10, quantes_ca_dia)
> m2
[1]  9  9 11  8  6
```

28. *PermsRec.*

Dissenyeu una funció `perm` que donat un nombre n emmagatzemi les permutacions dels n primers nombres en una matriu. La funció `perm(n)` ha d'utilitzar (de manera intel·ligent) `perm(n-1)`. En el cas de `perm(3)` obteniu:

```
> source("perm.R")
> P<-perm(3)
> P
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    1    3    2
[3,]    2    1    3
[4,]    2    3    1
[5,]    3    1    2
[6,]    3    2    1
```

Capítol 3

Lists

Objectiu: Introduir l'estructura `list` de R i totes les seves possibilitats com a estructura de dades que permet agrupar dades de diferents tipus.

3.1 Exercicis

1. Feu una funció que llegeixi de teclat un text conformat només per paraules i retorni un list que indica per a cada paraula en quines posicions del text està.

```
readlist ← function ()
{
    txt ← scan(what='character')
    wl ← list()
    for (i in 1:length(txt)) {
        wrd ← txt[i]
        wl[[wrd]] ← c(wl[[wrd]],i)
    }
    return (wl)
}
```

2. Feu una funció que rebi el list generat en l'exercici anterior i retorni un altre list amb el nombre de vegades que cada paraula apareix al text.

```
countinlist ← function (wlist)
{
    wcount ← list()
    noms ← names(wlist)
    for (i in 1:length(wlist)) {
        wrd ← noms[i]
   wcount[[wrd]] ← lenght(wlist[[wrd]])
    }
    return (wcount)
}
```

3. Feu una funció que rebi un vector de paraules i retorni un list amb el nombre de vegades que cada paraula apareix al vector.

```
countwords ← function (v)
{
    wcount ← list()
    for (i in 1:length(v)) {
        wrd ← v[i]
   if (! is.null(wcount[[wrd]])) {
       wcount[[wrd]] ← wcount[[wrd]] + 1
        }
        else {
            wcount[[wrd]] ← 1
        }
    }
    return (wcount)
}
```

4. *Compta paraules.* (Examen recuperació 13/14). Fes una funció que donat un vector de paraules (text), retorni un list que compta els cops que han aparescut en el text (en el vector) les paraules `"a"`, `"tot"` i `"que"`.

   Per exemple, si ens entra el vector de paraules següent:

   ```
   "Per" "a" "un" "cop" "que" "tinc" "a" "la" "meva" "mare" "a" "casa"
   "i" "puc" "disfrutar-la" "deixa'm" "que" "li" "dediqui" "tot" "el"
   "temps" "que" "pugui"
   ```

   Hauríem de retornar el list següent:

   `list (a=3, que=3, tot=1)`

   és a dir:

   ```
   $a
   [1] 3

   $que
   [1] 3

   $tot
   [1] 1
   ```

```
comptaparaules ← function(text) {
    lres ← list()
    for (i in 1:length(text)) {
        if ((text[i] == "a") || (text[i] == "que")
            || (text[i] == "tot")) {
            if (!is.null(lres[[i]])) {
                lres[[i]] ← lres[[i]] + 1
            }
            else {
                lres[[i]] ← 1
            }
```

```
11          }
12      }
13      return (lres)
14  }
```

5. Feu una funció que rebi un vector de paraules i retorni un list que compta tots els anagrames que hi ha al vector (per a cada conjunt de lletres tindrem quants anagrames d'aquest conjunt de lletres hi ha al vector).

   Una paraula és anagrama d'una altra si i només si, els caràcters que tenen totes dues paraules són els mateixos, encara que estiguin ordenats de forma diferent. Per exemple: roma - amor - omar - mora, són anagrames totes 4 entre sí, i arbol - labor - borla, també ho són.

6. Feu ara una funció que rebi un vector de paraules i retorni un list amb tots els anagrames que hi ha al vector (per a cada conjunt de lletres tindrem els anagrames d'aquest conjunt de lletres que hi ha al vector).

7. Feu una funció que donat un list que conté un diccionari on per a cada paraula tenim la seva descripció, retorni un list modificat on s'han eliminat totes les entrades del diccionari (paraules + descripció) que la paraula té menys de 5 lletres (podeu usar la funció nchar(string)).

```
1  eliminamenysde5 ← function(dicc) {
2      noms ← names(dicc)
3      for (i in 1:length(noms)) {
4          if (nchar(noms[i]) < 5) {
5              dicc[[noms[i]]] ← NULL
6          }
7      }
8      return (dicc)
9  }
```

8. Feu una funció que donat un list que conté un diccionari on per a cada paraula tenim la seva descripció, i donat també un vector de paraules, modifiqui el list de manera que si les paraules del vector estaven al list les elimini. Considereu que el diccionari no té paraules repetides.

9. *Sou anual.* (Examen parcial 14/15). Tenim dos lists, un amb informació dels mesos treballats a l'any per un treballador (`L1`) i l'altre amb el sou per mes que aquests treballadors cobren (`L2`).

   Es demana fer una funció que a partir dels dos lists `L1` i `L2` retorni un nou list `L3` amb el sou guanyat en tot l'any per cada treballador.

   Per exemple:

```
L1 <- list (maria=12, joan=10, emma=8)
L2 <- list(joan=800, maria=1000, emma=1200, kim=600)
```

   el resultat ha de ser:

```
list (joan=8000, maria=12000, emma=9600, kim=0)
```

és a dir:

```
$joan
[1] 800

$maria
[1] 1200

$emma
[1] 9600

$kim
[1] 0
```

Fixeu-vos que pot ser que un treballador no estigui en el list `L1` i sí en el `L2` (com el Kim), i això vol dir que està en nòmina però no ha treballat aquest any (resultat, cobra 0 euros aquest any), però **no us trobareu un treballador que estigui en L1 i no en L2**.

```
1  souAnual ← function (L1, L2)
2  {
3      L3 ← list()
4      for (name in names(L2)) {
5          if (!is.null(L1[[name]])) {
6              L3[[name]] ← L2[[name]] * L1[[name]]
7          }
8          else {
9              L3[[name]] ← 0
10         }
11     }
12     return (L3)
13 }
```

10. Feu una acció que rebi un list construït amb dos camps, un amb un vector amb les notes dels parcials d'una assignatura, i l'altre amb un vector amb les ponderacions de cada un, i escrigui per pantalla la informació de si l'alumne està suspès o aprovat (una cadena de caràcters que val "Suspès" o "Aprovat") i amb quina nota.

11. Feu una funció que rebi un list amb tres camps de tipus vector amb les notes dels parcials de tres assignatures respectivament i retorni un list amb un camp amb les mitjanes de les tres assignatures i un segon camp amb el nombre de notes de cada assignatura que estan per sobre de la mitjana de l'assignatura. *Nota: Podeu usar la funció mean(v) que calcula la mitjana d'un vector*

12. Sigui un list `eleccions` amb els resultats de les 5 últimes eleccions al Parlament de Catalunya on el primer camp és un vector amb els anys i els altres camps són vectors amb els resultats obtinguts per cadascun dels 5 primers partits (per aquest ordre, CIU, ERC, PSC, PP i ICV). Sempre tenint com a entrada el list `eleccions`, dissenyeu:

(a) Una funció que a partir del list `eleccions` retorni quin any el PP va obtenir el seu pitjor resultat i quin resultat va ser.

```
PP_pitjor ← function(llista){
   minPP ← eleccions$PP[1]
   any ← eleccions$Anys[1]
   for (i in 2:5){
      if(eleccions$PP[i] < minPP){
         minPP ← eleccions$PP[i]
         any ← eleccions$Anys[i]
      }
   }
   resultat ← c(minPP, any)
   return(resultat)
}

#resultat ← PP_pitjor(eleccions)
#cat("El PP va obtenir ", resultat[1], "escons
   l'any", resultat[2], "i aquest es el seu
   pitjor resultat \n")
```

(b) Una funció que a partir del list `eleccions` retorni quin any la diferència d'escons entre CIU i PSC va ser més gran i quina va ser aquesta diferència.

```
difCIU_PSC ← function(llista){
   dif ← 0
   for (i in 1:5){
      if(eleccions$CIU[i] - eleccions$PSC[i] >
         dif){
         dif ← eleccions$CIU[i] - eleccions$PSC
            [i]
         any ← eleccions$Anys[i]
      }
   }
   resultat ← c(dif, any)
   return(resultat)
}

#resultat ← difCIU_PSC(eleccions)
#cat("La diferencia mes gran d'escons entre CIU
    i PSC va ser de ", resultat[1], "escons i
   es va donar l'any", resultat[2], "\n")
```

(c) Una funció que a partir del list `eleccions` retorni quin partit va obtenir més escons l'any 2003 i quants escons va obtenir.

```
max_esc_2003 ← function(llista){
   max_esc ← llista[[2]][4]
   partit ← names(llista[2])
   for (i in 3:6){
      if (llista[[i]][4] > max_esc){
```

```
6            max_esc ← llista[[i]][4]
7            partit ← names(llista[i])
8        }
9    }
10   resultat ← c(max_esc, partit)
11 }
12
13 #resultat ← max_esc_2003(eleccions)
14 #cat("El partit que va obtenir mes escons (",
       resultat[1], ") l'any 2003 va ser", resultat
       [2], "\n")
```

13. *Guanys i pèrdues.* (Examen final 13/14). Fes una funció que donats dos lists, `guanys` i `perdues`, tals que tots dos tenen el format:

    clau: nom
    valor: quantitat de diners que ha guanyat/perdut

    (els valors de `guanys` són tots positius mentre que els valors de `perdues` són tots negatius) modifiqui el list `guanys` fent la "fusió" de tots dos lists, incloent tots els elements de `guanys` i els de `perdues` i en el cas d'elements que estan en tots dos fent la suma dels dos valors (guanys + perdues).

    Per exemple:

    ```
    guanys <- list (pepet=3000, maria=2000, joan=1000)
    perdues <- list(joan=-500, maria=-1000, kim=-400)
    ```

    el resultat ha de ser:

    ```
    list (pepet=3000, maria=1000, joan=500, kim=-400)
    ```

    és a dir:

    ```
    $pepet
    [1] 3000

    $maria
    [1] 1000

    $joan
    [1] 500

    $kim
    [1] -400
    ```

14. Sigui un list `ed_al` amb dues components de tipus vector que contenen informació sobre l'edat (primera component) i l'alçada (segona component) de 20 persones. Sempre rebent com a entrada el list `ed_al`, dissenyeu les següents funcions:

    (a) Una funció que donat el list `ed_al` retorni quina és l'alçada de la persona més alta.

```
persona_mes_alta ← function(llista){
   alc_max ← llista$alc[1]
   for(i in 1:length(llista$edat)){
      if (llista$alc[i] > alc_max){
         alc_max ← llista$alc[i]
      }
   }
   return(alc_max)
}

# cat("L'çalada de la persona mes alta es",
   persona_mes_alta(ed_al), "\n")
```

(b) Una funció que donat el list `ed_al` retorni quina edat té la persona més baixa.

```
ed_mes_baixa ← function(llista){
   alc_min ← llista$alc[1]
   ind ← 1
   for(i in 1:length(llista$edat)){
      if (llista$alc[i] < alc_min){
         alc_min ← llista$alc[i]
         edat ← llista$edat[i]
      }
   }
   return(edat)
}

# cat("L'edat de la persona mes baixa es",
   ed_mes_baixa(ed_al), "\n")
```

(c) Una funció que donat el list `ed_al` retorni quantes persones tenen edats compreses entre 30 i 40 anys (inclosos).

```

trenta_quaranta ← function(llista){
   tot ← 0
   for (i in 1:length(ed_al$edat)){
      if(ed_al$edat[i] %in% 30:40){
         tot ← tot +1
      }
   }
   return(tot)
}

# cat("Hi ha ", trenta_quaranta(ed_al), "
   persones amb edats entre 30 i 40 anys \n")
```

15. *Modificant lists.* (Examen final 13/14). Tenim la següent funció de lists ja implementada:

```
ex2 <- function (L1, L2)
{
    for (n in names(L2)) {
        if (!is.null(L1[[n]])) {
            L2[[n]] <- L2[[n]] * L1[[n]] / 100
        }
        else {
            L2[[n]] <- NULL
        }
    }
    return (L2)
}
```

Si els dos lists d'entrada d'aquesta funció són els següents:

```
L1 <- list (Maria=20,Nuria=10,Pep=10,Miquel=5,Jana=15,Joan=15,Marc=10)
L2 <- list (Joan=500,Pep=340,Maria=250,Miquel=700,Nuria=560,Isabel=670,
            Marc=930,Axel=480)
```

quina serà la sortida de la funció?

```
1  L2 ← list (Joan=75,Pep=34,Maria=50,Miquel=35,
2             Nuria=56,Marc=93)
```

16. *Fusió.* (Examen parcial 13/14). Fes una funció que donats dos lists, L1 i L2, tals que tots dos tenen el format:

clau: nom
valor: nombre de vegades que ha anat al cinema

modifiqui el list L1 fent la "fusió" de tots dos lists, incloent tots els elements de L1 i L2 i en el cas d'elements que estan en tots dos fent la suma dels dos valors.

Per exemple:

```
L1 <- list (pepet=3, maria=2, joan=1)
L2 <- list(joan=5, maria=1, kim=4)
```

el resultat ha de ser:

```
list (pepet=3, maria=3, joan=6, kim=4)
```

és a dir:

```
$pepet
[1] 3

$maria
[1] 3
```

```
$joan
[1] 6

$kim
[1] 4
```

```
fusiolists ← function(l1, l2) {
    for (i in names(l2)) {
        if (!is.null(l1[[i]])) {
            l1[[i]] ← l1[[i]] + l2[[i]]
        }
        else {
            l1[[i]] ← l2[[i]]
        }
    }
    return (l1)
}
```