Nathalie Vialaneix
Année 2019/2020

M2 in Statistics & Econometrics

Graph mining

Lesson 1 - Introduction to graphs and networks

This worksheet illustrates the use of the R package **igraph** to create and manipulate graphs. It shows basic data mining tools that are provided in this package to obtain an overview of the graph main properties and to extract its most important vertices. The packages **RColorBrewer** and **ggplot2** will also be used in this worksheet. Start loading all the packages with:

```
library(igraph)
library(RColorBrewer)
library(ggplot2)
```

The data used to illustrate this work can be found at `http://www.nathalievialaneix.eu/doc/zip/data_M2SE.zip` (for GOT and FB networks; once uncompressed you obtain three data files, as described in the lesson and two README files that describe the data) and at `http://www.nathalievialaneix.eu/doc/txt/fbnet-el-2015.txt` and `http://www.nathalievialaneix.eu/doc/txt/fbnet-name-2015.txt` for (respectively) the edge list and the initials of the vertices (NVV network). Load all these files and put them in a subdirectory called `data`. Create your R script `lesson1.R` in another subdirectory (located in the same place than `data`) called `RLib`.

Exercice 1 Create a graph

1. The simplest way to create a graph with **igraph** is to create them from an edge list (function `graph_from_edgelist`) or from their adjacency matrix (function `graph_from_adjacency_matrix`).

   (a) *Create a graph from an edgelist*: load the NVV edge list into a matrix called `nvv_el` and create your first `igraph` object, called `nvv_net` is created with (read the help of the function `graph_from_edgelist` for details about the options):

```
nvv_el <- read.table("../data/fbnet-el-2015.txt", header = FALSE)
nvv_net <- graph_from_edgelist(as.matrix(nvv_el), directed = FALSE)
nvv_net

## IGRAPH 6548b82 U--- 152 551 --
## + edges from 6548b82:
##  [1]  1-- 11  1-- 41  1-- 52  1-- 69  1-- 74  1-- 75  1-- 77  1-- 78
##  [9]  1-- 81  1-- 83  1-- 86  1-- 87  1-- 90  1-- 99  1--112  1--116
## [17]  1--151  2-- 33  2-- 63  2-- 73  3--  4  3-- 14  3-- 16  3-- 39
## [25]  4-- 14  4-- 16  5-- 23  5-- 46  5-- 73  5--119  5--134  5--150
## [33]  6-- 33  6-- 51  7-- 33  7-- 57  7-- 76  8-- 11  8-- 17  8-- 18
## [41]  8-- 22  8-- 38  8-- 41  8-- 43  8-- 52  8-- 62  8-- 68  8-- 69
## [49]  8-- 72  8-- 74  8-- 79  8-- 80  8-- 81  8-- 85  8-- 86  8-- 87
## [57]  8-- 90  8-- 92  8-- 93  8--100  8--101  8--102  8--105  8--112
## [65]  8--118  8--121  8--122  8--126 10-- 12 10-- 13 10--123 11-- 17
## + ... omitted several edges
```

   The output reads: the `igraph` object is **U**ndirected, its vertices are not named (otherwise, the `U` would have been followed by a `N`), its edges are not weighted (otherwise, the third place would have been a `W`) and it is not bipartite (otherwise, the last place would have been a `B`). It has 152 vertices and 551 edges.
   More information about the output can be found with `help(igraph)`.

(b) *Create a graph from an adjacency matrix*: load the FB adjacency matrix into a matrix called `fb_ajd` and create your second `igraph` object, called `fb_net` with:

```
fb_adj <- read.table("../data/Amherst41.adjacency", sep = "\t", header = FALSE)
fb_net <- graph_from_adjacency_matrix(as.matrix(fb_adj), mode = "undirected")
fb_net
```

```
## IGRAPH e3da9e7 UN-- 2235 90954 --
## + attr: name (v/c)
## + edges from e3da9e7 (vertex names):
##  [1] V1--V254  V1--V358  V1--V385  V1--V760  V1--V779  V1--V842  V1--V1121
##  [8] V1--V1185 V1--V1308 V1--V1311 V1--V1323 V1--V1329 V1--V1389 V1--V1500
## [15] V1--V1515 V1--V1545 V1--V1561 V1--V1613 V1--V1623 V1--V1640 V1--V1835
## [22] V1--V1902 V1--V2011 V1--V2030 V1--V2056 V2--V3    V2--V28   V2--V45
## [29] V2--V48   V2--V53   V2--V64   V2--V70   V2--V79   V2--V91   V2--V96
## [36] V2--V153  V2--V179  V2--V222  V2--V253  V2--V263  V2--V265  V2--V327
## [43] V2--V337  V2--V371  V2--V385  V2--V452  V2--V476  V2--V578  V2--V597
## [50] V2--V609  V2--V649  V2--V685  V2--V695  V2--V706  V2--V712  V2--V793
## + ... omitted several edges
```

How do you interpret the last output? What differences do you see with the previous one? The output `attr` indicate the graph attributes: this graph has an attribute called `name`, which is related to its vertices (`v`). The vertices are named `V1`, `V2`, ... instead of just using numbers: these names correspond to the column names given of `fb_adj` that were automatically given by the function `read.table`.

(c) *Do it yourself!* Create the (unweighted) GOT network (after the data have been imported in a data frame `got`):

```
## IGRAPH bbbf524 UN-- 107 352 --
## + attr: name (v/c)
## + edges from bbbf524 (vertex names):
##  [1] Aemon  --Grenn     Aemon  --Samwell   Aerys  --Jaime
##  [4] Aerys  --Robert    Aerys  --Tyrion    Aerys  --Tywin
##  [7] Alliser--Mance     Amory  --Oberyn    Arya   --Anguy
## [10] Arya   --Beric     Arya   --Bran      Arya   --Brynden
## [13] Arya   --Cersei    Arya   --Gendry    Arya   --Gregor
## [16] Jaime  --Arya      Arya   --Joffrey   Arya   --Jon
## [19] Arya   --Rickon    Robert --Arya      Arya   --Roose
## [22] Arya   --Sandor    Arya   --Thoros    Tyrion --Arya
## + ... omitted several edges
```

Other graph constructors are described in `help(make_graph)`.

2. The functions `vcount` and `ecount` can be used to find the number of vertices and edges of an `igraph` object:

```
vcount(nvv_net); ecount(nvv_net)
```

```
## [1] 152
## [1] 551
```

3. Vertices and edges themselves are obtained with the functions `V` and `E`:

```
head(V(got_net))
```

```
## + 6/107 vertices, named, from bbbf524:
## [1] Aemon   Grenn   Samwell Aerys   Jaime   Robert
```

```
head(E(got_net))
```

```
## + 6/352 edges from bbbf524 (vertex names):
## [1] Aemon--Grenn   Aemon--Samwell Aerys--Jaime   Aerys--Robert
## [5] Aerys--Tyrion  Aerys--Tywin
```

Their attributes are accessed with $followed by the name of the attributes. For instance:

```
head(V(fb_net)$name)

## [1] "V1" "V2" "V3" "V4" "V5" "V6"
```

(a) *add an attribute to the vertices*: load the NVV initials and add it to the vertex attribute `name`:

```
nvv_info <- read.table("../data/fbnet-name-2015.txt", header = FALSE,
                       stringsAsFactors = FALSE)
V(nvv_net)$name <- nvv_info$V1
nvv_net

## IGRAPH 6548b82 UN-- 152 551 --
## + attr: name (v/c)
## + edges from 6548b82 (vertex names):
##  [1] J.M--C.T   J.M--I.G   J.M--C.T   J.M--A.R   J.M--V.G   J.M--S.D
##  [7] J.M--N.L.C J.M--K.L   J.M--A.B   J.M--N.P   J.M--A.L   J.M--L.A
## [13] J.M--E.P   J.M--R.F   J.M--L.R.P J.M--C.H   J.M--L.M   C.L--M.D
## [19] C.L--V.T   C.L--D.L   C.R--J.R   C.R--F.A   C.R--S.T   C.R--P.C
## [25] J.R--F.A   J.R--S.T   M.V--F.R   M.V--L.T   M.V--D.L   M.V--B.S
## [31] M.V--C.D   M.V--C.N   K.A--M.D   K.A--J.D   f.p--M.D   f.p--M.C
## [37] f.p--P.L   S.L--C.T   S.L--N.P   S.L--C.C   S.L--N.T   S.L--E.P
## [43] S.L--I.G   S.L--M.P   S.L--C.T   S.L--M.C   S.L--E.L   S.L--A.R
## + ... omitted several edges
```

(b) *add an attribute to the edges*: use the third column of the data frame `got` to obtain a weighted network:

```
E(got_net)$weight <- got[ ,3]
got_net

## IGRAPH bbbf524 UNW- 107 352 --
## + attr: name (v/c), weight (e/n)
## + edges from bbbf524 (vertex names):
##  [1] Aemon   --Grenn     Aemon  --Samwell    Aerys   --Jaime
##  [4] Aerys   --Robert    Aerys  --Tyrion     Aerys   --Tywin
##  [7] Alliser--Mance      Amory  --Oberyn     Arya    --Anguy
## [10] Arya    --Beric     Arya   --Bran       Arya    --Brynden
## [13] Arya    --Cersei    Arya   --Gendry     Arya    --Gregor
## [16] Jaime   --Arya      Arya   --Joffrey    Arya    --Jon
## [19] Arya    --Rickon    Robert --Arya       Arya    --Roose
## [22] Arya    --Sandor    Arya   --Thoros     Tyrion --Arya
## + ... omitted several edges
```

(c) *Do it yourself!* Import the data in the file `Amherst41.info` a data frame `fb_info` and use the second column (corresponding to gender) to create a vertex attribute named `color` equal to `"pink"` if the value is 2, to `"lightblue"` if the value is 1 and to `"black"` if the value is 0.

```
## IGRAPH e3da9e7 UN-- 2235 90954 --
## + attr: name (v/c), color (v/c)
## + edges from e3da9e7 (vertex names):
##  [1] V1--V254  V1--V358  V1--V385  V1--V760  V1--V779  V1--V842  V1--V1121
##  [8] V1--V1185 V1--V1308 V1--V1311 V1--V1323 V1--V1329 V1--V1389 V1--V1500
## [15] V1--V1515 V1--V1545 V1--V1561 V1--V1613 V1--V1623 V1--V1640 V1--V1835
## [22] V1--V1902 V1--V2011 V1--V2030 V1--V2056 V2--V3    V2--V28   V2--V45
## [29] V2--V48   V2--V53   V2--V64   V2--V70   V2--V79   V2--V91   V2--V96
## [36] V2--V153  V2--V179  V2--V222  V2--V253  V2--V263  V2--V265  V2--V327
## [43] V2--V337  V2--V371  V2--V385  V2--V452  V2--V476  V2--V578  V2--V597
## [50] V2--V609  V2--V649  V2--V685  V2--V695  V2--V706  V2--V712  V2--V793
## + ... omitted several edges
```

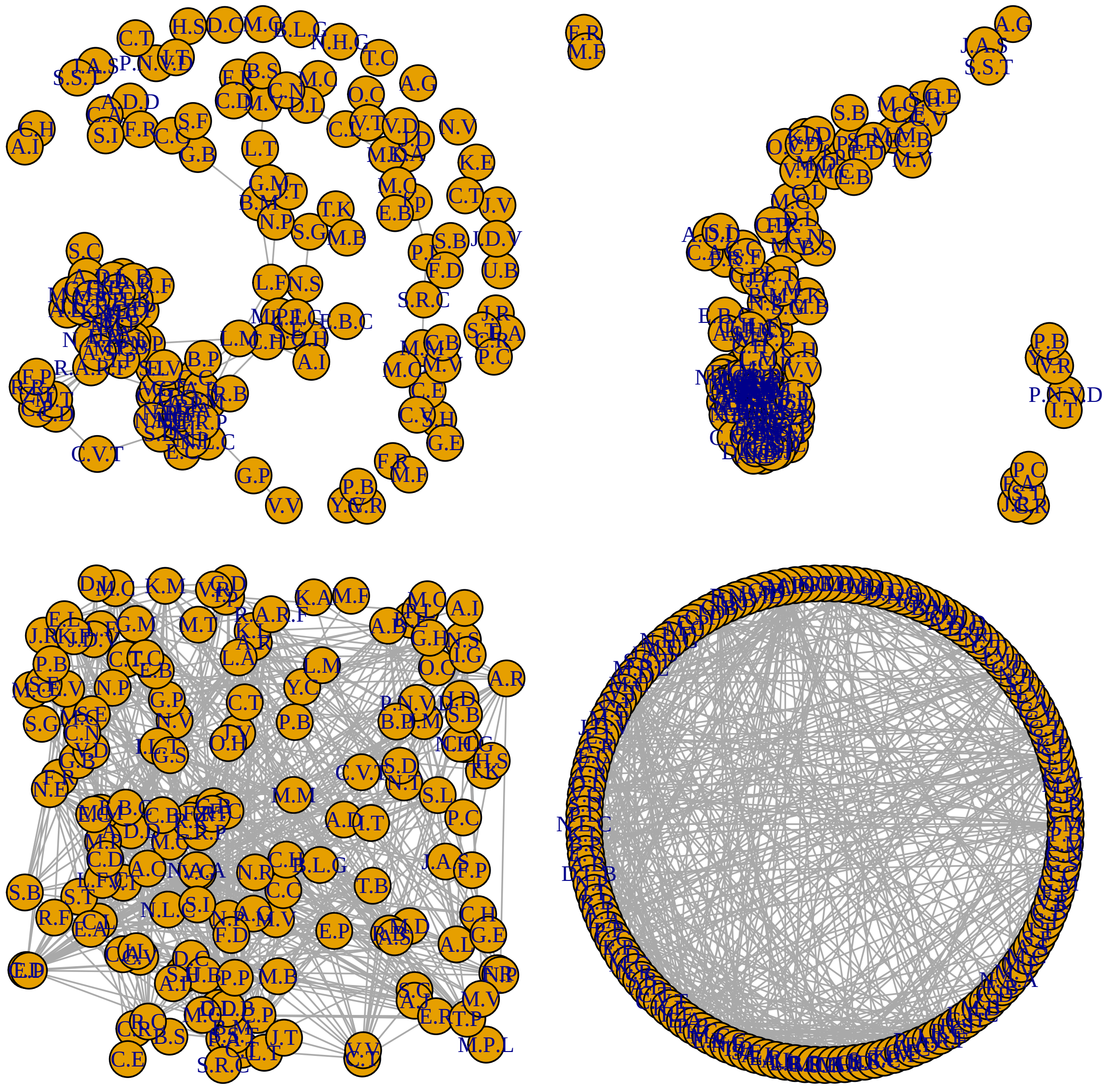
Using the `igraph` object, how many boys/girls do we have in this network?

```
##
##      black lightblue      pink
##        203      1015      1017
```

4. Different layouts for displaying the graph are available in **igraph**. They are named `layout_...` and are described in `help(layout, package = igraph)`. They produce a 2 (or 3) dimensional matrix with the vertices coordinates or can be used directly in combination with the function `plot`.

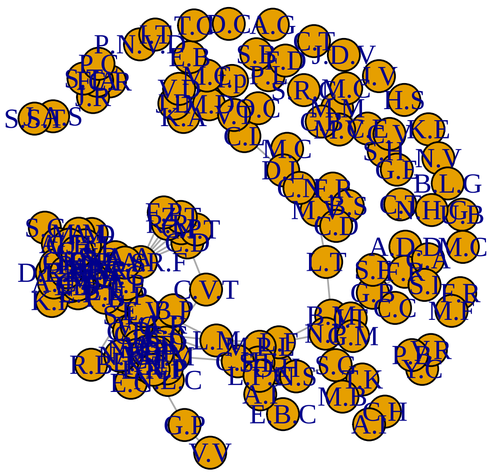
   (a) The most useful layouts are `layout_with_fr` (Fruchterman and Reingold layout), `layout_with_kk` (Kamada Kawai layout, another force directed algorithm), `layout_randomly` (random layout) or `layout_in_circle` (circular layout):

```
par(mfrow = c(2,2))
par(mar = rep(0,4))
plot(nvv_net, layout = layout_with_fr)
par(mar = rep(0,4))
plot(nvv_net, layout = layout_with_kk)
par(mar = rep(0,4))
plot(nvv_net, layout = layout_randomly)
par(mar = rep(0,4))
plot(nvv_net, layout = layout_in_circle)
```



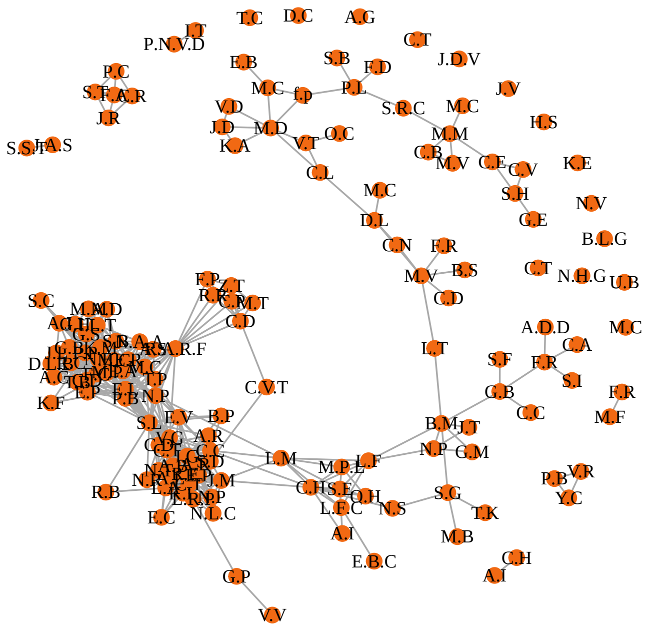
(b) `layout_nicely` should choose the most appropriate layout for a given graph. However, when running several times most layout functions, different layouts are produced due to the randomness included in the layout generating process. This can be fixed by using a random seed before calling the layout function and/or storing the layout in a graph attribute called `layout`, which would be automatically used by the function `plot` unlike said:

```
set.seed(1600)
nvv_net$layout <- layout_nicely(nvv_net)
nvv_net

## IGRAPH 6548b82 UN-- 152 551 --
## + attr: layout (g/n), name (v/c)
## + edges from 6548b82 (vertex names):
##  [1] J.M--C.T   J.M--I.G   J.M--C.T   J.M--A.R   J.M--V.G   J.M--S.D
##  [7] J.M--N.L.C J.M--K.L   J.M--A.B   J.M--N.P   J.M--A.L   J.M--L.A
## [13] J.M--E.P   J.M--R.F   J.M--L.R.P J.M--C.H   J.M--L.M   C.L--M.D
## [19] C.L--V.T   C.L--D.L   C.R--J.R   C.R--F.A   C.R--S.T   C.R--P.C
## [25] J.R--F.A   J.R--S.T   M.V--F.R   M.V--L.T   M.V--D.L   M.V--B.S
## [31] M.V--C.D   M.V--C.N   K.A--M.D   K.A--J.D   f.p--M.D   f.p--M.C
## [37] f.p--P.L   S.L--C.T   S.L--N.P   S.L--C.C   S.L--N.T   S.L--E.P
## [43] S.L--I.G   S.L--M.P   S.L--C.T   S.L--M.C   S.L--E.L   S.L--A.R
## + ... omitted several edges

par(mar = rep(0,4))
plot(nvv_net)
```

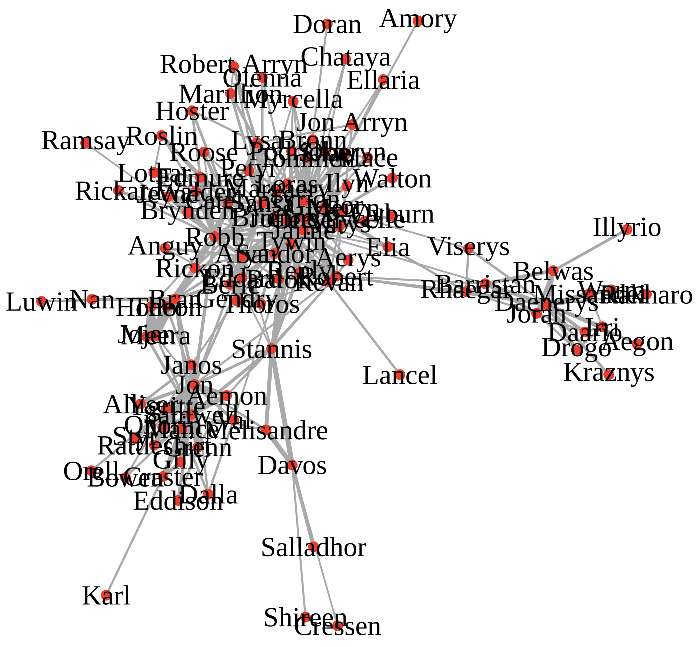


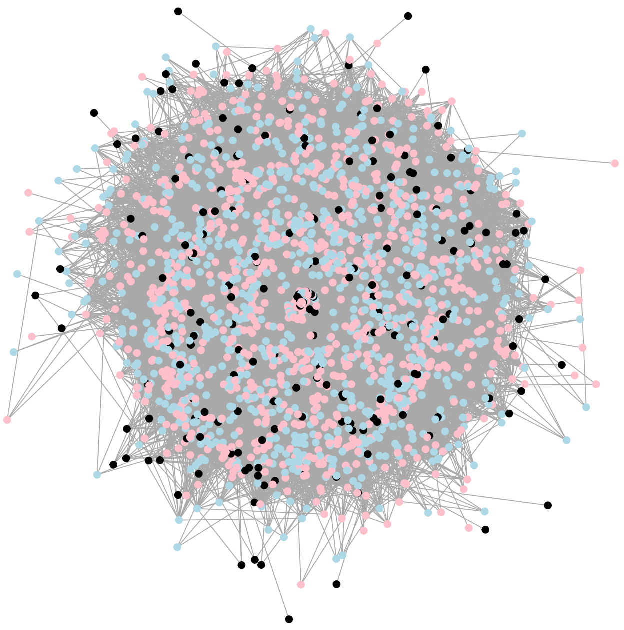
(c) Vertices colors, sizes, labels, ... are handled with arguments `vertex.color`, `vertex.size`, `vertex.label`, ... or by setting a corresponding attribute in the graph (`V(nvv_net)$color` for instance). Edges colors, width, type are handled similarly. For instance, a more complex graph representation is obtained with:

```
nvv_col <- brewer.pal(9, "Oranges")[6]
par(mar = rep(0,4))
plot(nvv_net, vertex.size = 5, vertex.color = nvv_col,
     vertex.frame.color = nvv_col, vertex.label.color = "black",
     vertex.label.cex = 0.7)
```



(d) *Do it yourself!* Use the plotting options to obtain the following representation of the networks (for GOT, the layout is the default layout, the color is `got_col <- brewer.pal(9, "Reds")[6]`, the edge width are equal to the square root of the edge weights divided by the maximum weight and multiplied by 5; for FB, the layout is `layout_with_kk`):





More information about **igraph** plotting possibilities are provided with `help(igraph.plotting)`

5. (a) The function `is_connected` indicates if the graph is connected:

```
is_connected(nvv_net); is_connected(got_net)

## [1] FALSE
## [1] TRUE
```

(b) For non connected graphs, the different connected components can be found with `clusters`:

```
nvv_cc <- clusters(nvv_net)
nvv_cc

## $membership
##     J.M     C.L     C.R     J.R     M.V     K.A     f.p     S.L     K.E
##       1       1       2       2       1       1       1       1       3
##     S.H     C.T     C.E     C.V     F.A     N.V     S.T     N.P     C.C
##       1       1       1       1       2       4       2       1       1
##     C.H     J.V     C.T     N.T     F.R     D.C   A.D.D     O.H     F.R
##       5       6       7       1       1       8       1       1       1
##     B.M     C.A     F.R     N.S   B.L.G     M.D     O.C     C.D     G.P
##       1       1       9       1      10       1       1       1       1
##     M.M     E.P     P.C     S.G     I.G     A.G     M.P     S.C     G.B
##       1       1       2       1       1      11       1       1       1
##     L.T P.N.V.D     U.B     T.K     G.B     J.D     C.T     C.P     M.C
##       1      12      13       1       1       1       1       1       1
##     L.F   N.H.G     M.C   J.A.S   S.S.T   M.P.L     C.D     M.C     V.T
##       1      14       1      15      15       1       1       1       1
##     N.P     Y.C     M.T   J.D.V     E.L     A.R     E.C     V.V     A.R
##       1      16       1      17       1       1       1       1       1
##     D.L     V.G     S.D     P.L   N.L.C     K.L     B.P     E.V     A.B
##       1       1       1       1       1       1       1       1       1
##   D.L.B     N.P     J.T     R.B     A.L     L.A     P.P     C.C     E.P
##       1       1       1       1       1       1       1       1       1
##     V.D     M.C     N.R     S.B     A.D     C.T   C.V.T     K.F     R.F
##       1       1       1       1       1      18       1       1       1
##     E.A     P.B     F.C     M.C   E.B.C     N.E     T.B     S.E     A.J
##       1       1       1       1       1       1       1       1       1
##     E.R     A.I     J.L   L.R.P     R.R     M.M     S.B     C.H     A.I
##       1       5       1       1       1       1       1       1       1
##     C.D     B.S   S.R.C     T.P     K.M     G.E     A.C     F.D R.A.R.F
##       1       1       1       1       1       1       1       1       1
##   I.L.T     R.C     H.S     G.H   L.F.C     M.B     Z.T     C.D     E.B
##       1       1      19       1       1       1       1       1       1
##   N.A.A     M.V     M.F     M.C     A.S     S.F     I.T     S.I     C.B
##       1       1       9      20       1       1      12       1       1
##     V.R     F.P     G.M     T.C     G.S     C.N     L.M     P.B
##      16       1       1      21       1       1       1      16
##
## $csize
##  [1] 122   5   1   1   2   1   1   1   2   1   1   2   1   1   2   3   1
## [18]   1   1   1   1
##
## $no
## [1] 21
```

The outputs of this function are: `membership` that gives a number for each vertex, each number corresponding to a connected component; `csize` that provides the size of the connected components (number of vertices that are inside); `no` that is the number of connected components. What are the vertices included in the largest connected component? Store them in a vector called `v_lcc`:

```
##     J.M     C.L     M.V     K.A     f.p     S.L     S.H     C.T     C.E
##       1       2       5       6       7       8      10      11      12
##     C.V     N.P     C.C     N.T     F.R   A.D.D     O.H     F.R     B.M
##      13      17      18      22      23      25      26      27      28
##     C.A     N.S     M.D     O.C     C.D     G.P     M.M     E.P     S.G
##      29      31      33      34      35      36      37      38      40
##     I.G     M.P     S.C     G.B     L.T     T.K     G.B     J.D     C.T
##      41      43      44      45      46      49      50      51      52
##     C.P     M.C     L.F     M.C   M.P.L     C.D     M.C     V.T     N.P
##      53      54      55      57      60      61      62      63      64
##     M.T     E.L     A.R     E.C     V.V     A.R     D.L     V.G     S.D
##      66      68      69      70      71      72      73      74      75
##     P.L   N.L.C     K.L     B.P     E.V     A.B   D.L.B     N.P     J.T
##      76      77      78      79      80      81      82      83      84
##     R.B     A.L     L.A     P.P     C.C     E.P     V.D     M.C     N.R
##      85      86      87      88      89      90      91      92      93
##     S.B     A.D   C.V.T     K.F     R.F     E.A     P.B     F.C     M.C
##      94      95      97      98      99     100     101     102     103
##   E.B.C     N.E     T.B     S.E     A.J     E.R     J.L   L.R.P     R.R
##     104     105     106     107     108     109     111     112     113
##     M.M     S.B     C.H     A.I     C.D     B.S   S.R.C     T.P     K.M
##     114     115     116     117     118     119     120     121     122
##     G.E     A.C     F.D R.A.R.F   I.L.T     R.C     G.H   L.F.C     M.B
##     123     124     125     126     127     128     130     131     132
##     Z.T     C.D     E.B   N.A.A     M.V     A.S     S.F     S.I     C.B
##     133     134     135     136     137     140     141     143     144
##     F.P     G.M     G.S     C.N     L.M
##     146     147     149     150     151
```

(c) A subgraph induced by a set of vertices can be obtained with the function `induced_subgraph`. Hence, the largest connected component of NVV is:

```
nvv_lcc <- induced_subgraph(nvv_net, v_lcc)
nvv_lcc

## IGRAPH 30b8309 UN-- 122 535 --
## + attr: layout (g/n), name (v/c)
## + edges from 30b8309 (vertex names):
```

```
##  [1] J.M--C.T   J.M--I.G   J.M--C.T   J.M--A.R   J.M--V.G   J.M--S.D
##  [7] J.M--N.L.C J.M--K.L   J.M--A.B   J.M--N.P   J.M--A.L   J.M--L.A
## [13] J.M--E.P   J.M--R.F   J.M--L.R.P J.M--C.H   J.M--L.M   C.L--M.D
## [19] C.L--V.T   C.L--D.L   M.V--F.R   M.V--L.T   M.V--D.L   M.V--B.S
## [25] M.V--C.D   M.V--C.N   K.A--M.D   K.A--J.D   f.p--M.D   f.p--M.C
## [31] f.p--P.L   S.L--C.T   S.L--N.P   S.L--C.C   S.L--N.T   S.L--E.P
## [37] S.L--I.G   S.L--M.P   S.L--C.T   S.L--M.C   S.L--E.L   S.L--A.R
## [43] S.L--A.R   S.L--V.G   S.L--B.P   S.L--E.V   S.L--A.B   S.L--R.B
## + ... omitted several edges
```

(d) *Exercise:* Is FB network connected? Use the function `delete_edges` to delete 20,000 edges randomly chosen. Store the result in a graph denoted by `fb_net2`. Is this graph connected? If not, extract its largest connected component. How large is it?

```
## [1] "connected: FALSE"
## IGRAPH 1eda054 UN-- 2225 70954 --
## + attr: name (v/c), color (v/c)
## + edges from 1eda054 (vertex names):
##  [1] V1--V254  V1--V358  V1--V385  V1--V760  V1--V842  V1--V1121 V1--V1185
##  [8] V1--V1308 V1--V1311 V1--V1329 V1--V1500 V1--V1515 V1--V1561 V1--V1640
## [15] V1--V1835 V1--V2011 V1--V2030 V1--V2056 V2--V28   V2--V45   V2--V48
## [22] V2--V53   V2--V64   V2--V96   V2--V179  V2--V253  V2--V265  V2--V327
## [29] V2--V371  V2--V385  V2--V452  V2--V476  V2--V578  V2--V597  V2--V649
## [36] V2--V695  V2--V712  V2--V793  V2--V845  V2--V861  V2--V881  V2--V915
## [43] V2--V967  V2--V971  V2--V1061 V2--V1154 V2--V1259 V2--V1263 V2--V1300
## [50] V2--V1423 V2--V1482 V2--V1545 V2--V1557 V2--V1561 V2--V1596 V2--V1600
## + ... omitted several edges
```

In my case, 99.6% of the original vertices are included in the largest connected component.

6. `igraph` objects can be exported in other formats with the `write_graph` function. For instance,

```
write_graph(nvv_lcc, file = "../results/nvv_lcc.graphml", format = "graphml")
```

exports the NVV largest connected component into a `graphml` file. Vertices and edges attributes are included in this file, which can be read (for instance) with software for interactive manipulation of networks like Gephi https://gephi.org.
In addition, the edge list (as a 2-column matrix) or the adjacency matrix can be retrieved from the `igraph` object with the functions `as_edgelist` and `as_adj`:

```
head(as_edgelist(nvv_lcc)); as_adj(nvv_lcc)[1:10,1:10]

##      [,1]  [,2]
## [1,] "J.M" "C.T"
## [2,] "J.M" "I.G"
## [3,] "J.M" "C.T"
## [4,] "J.M" "A.R"
## [5,] "J.M" "V.G"
## [6,] "J.M" "S.D"
## 10 x 10 sparse Matrix of class "dgCMatrix"

##    [[ suppressing 10 column names 'J.M', 'C.L', 'M.V' ...  ]]

##
## J.M . . . . . . . 1 . .
## C.L . . . . . . . . . .
## M.V . . . . . . . . . .
## K.A . . . . . . . . . .
## f.p . . . . . . . . . .
## S.L . . . . . . . 1 . .
## S.H . . . . . . . . 1 1
## C.T 1 . . . . 1 . . . .
## C.E . . . . . . 1 . . 1
## C.V . . . . . . 1 . 1 .
```

The adjacency matrix is usually returned as a sparse matrix. If standard matrix is needed, it is obtained with:

```
as_adj(nvv_lcc, sparse = FALSE)[1:10,1:10]

##     J.M C.L M.V K.A f.p S.L S.H C.T C.E C.V
## J.M   0   0   0   0   0   0   0   1   0   0
## C.L   0   0   0   0   0   0   0   0   0   0
## M.V   0   0   0   0   0   0   0   0   0   0
## K.A   0   0   0   0   0   0   0   0   0   0
## f.p   0   0   0   0   0   0   0   0   0   0
## S.L   0   0   0   0   0   0   0   1   0   0
## S.H   0   0   0   0   0   0   0   0   1   1
## C.T   1   0   0   0   0   1   0   0   0   0
## C.E   0   0   0   0   0   0   1   0   0   1
## C.V   0   0   0   0   0   0   1   0   1   0
```

Exercice 2 Global characteristics

1. Density, transitivity, diameter, radius, girth and cohesion of a graph are found with the functions `edge_density`, `transitivity`, `diameter`, `radius`, `girth` and `cohesion`. Use these functions to fill a data frame with the main characteristics for each network. The function `diameter` has a weighted and an unweigthed versions, depending on whether the shortest path lengths are calculated taking weights into account or not. Use the unweighted version that makes more sense in this context (because a large weight means a close connexion and not the opposite).

	density	transitivity	diameter	radius	girth	cohesion
NVV	0.048	0.562	18	0	3	0
NVV LCC	0.072	0.560	18	9	3	1
GOT	0.062	0.329	6	3	3	1
FB	0.036	0.233	7	4	3	1

2. The shortest paths between two vertices in a graph can be found with the function `shortest_paths`:

```
shortest_paths(nvv_lcc, from = "S.L", to = "B.M")

## $vpath
## $vpath[[1]]
## + 5/122 vertices, named, from 30b8309:
## [1] S.L N.P L.M L.F B.M
##
##
## $epath
## NULL
##
## $predecessors
## NULL
##
## $inbound_edges
## NULL
```

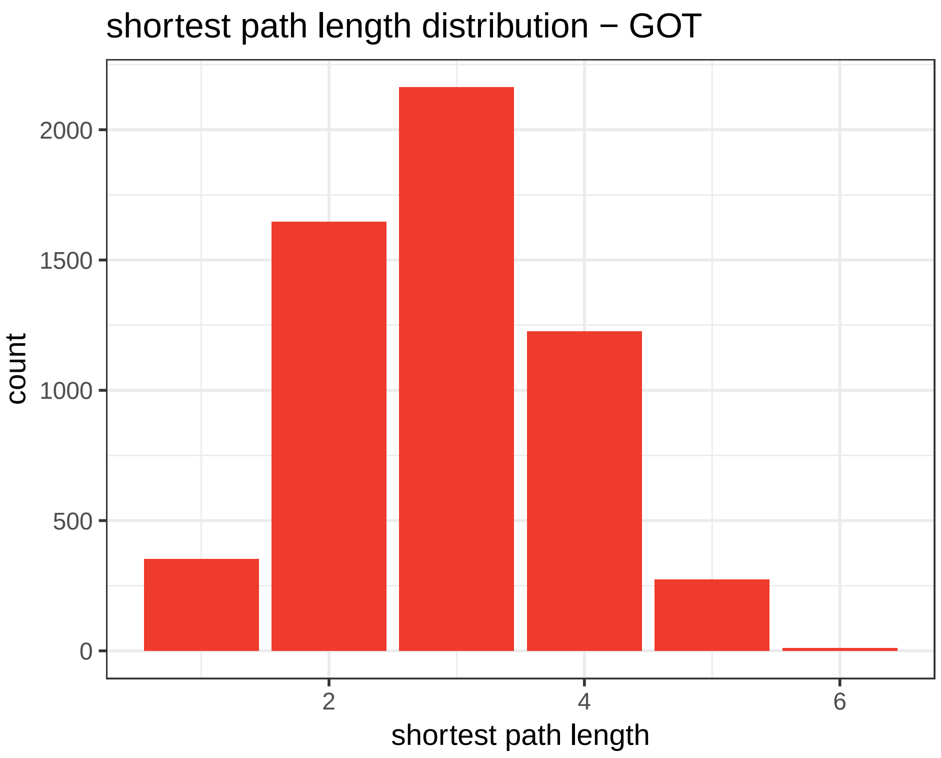
All shortest paths from a given vertex are obtained with `all_shortest_paths`. Find the shortest path between Jon and Tyrion in GOT network:

```
shortest_paths(got_net, from = "Jon", to = "Tyrion")$vpath

## [[1]]
## + 3/107 vertices, named, from bbbf524:
## [1] Jon    Arya   Tyrion
```

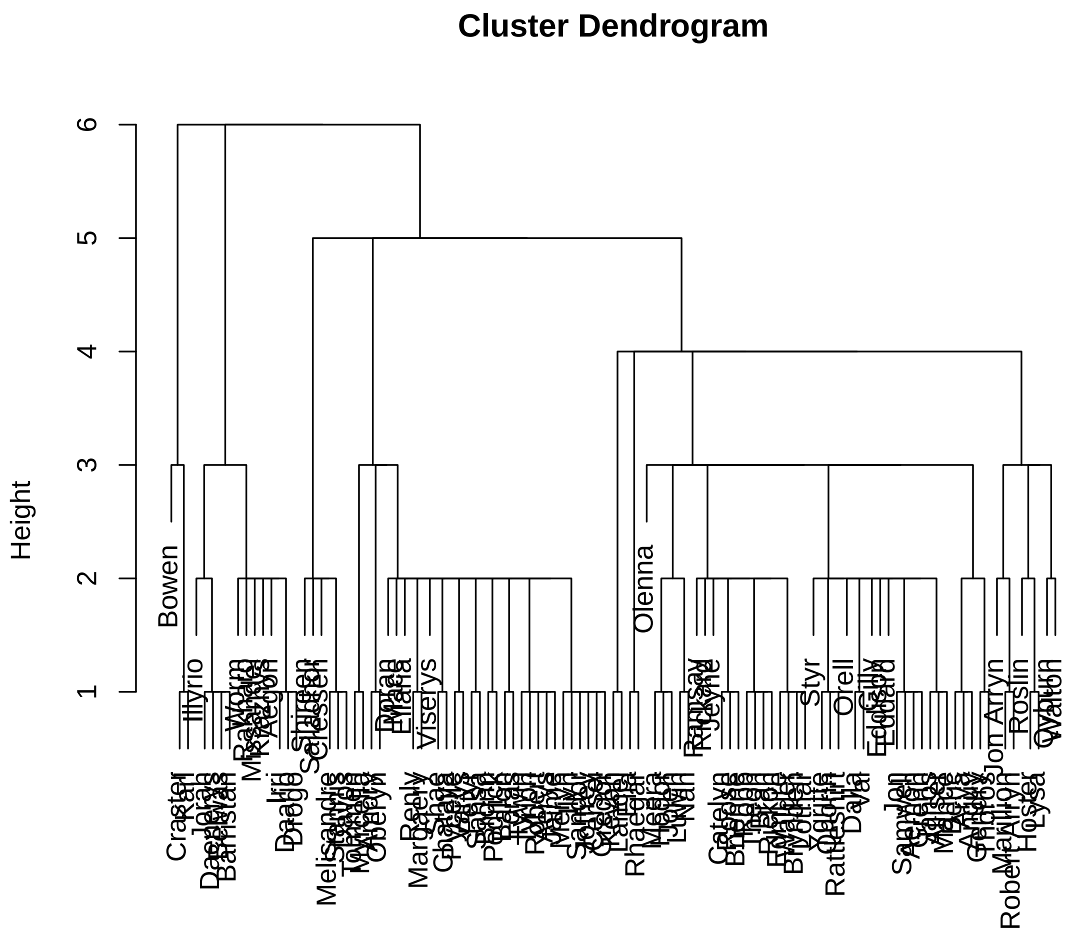
3. Shortest path lengths are obtained with `distances`. This function uses weights by default. For weighted networks, unweigted shortest path lenghts are computed using the option `algorithm = "unweighted"`. The result is a matrix of size $n \times n$.

   - The unweighted shortest path lengths for GOT are computed and their distribution is obtained with:

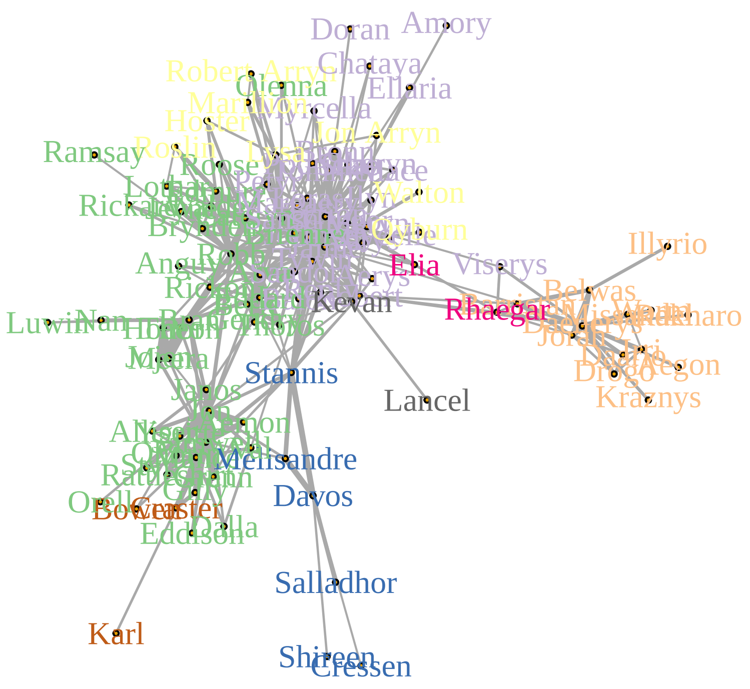
     ```
     sp_got <- distances(got_net, algorithm = "unweighted")

     ## Warning in distances(got_net, algorithm = "unweighted"):  Unweighted algorithm
     chosen, weights ignored

     sp_got <- sp_got[upper.tri(sp_got)]
     df <- data.frame(sp = sp_got)
     p <- ggplot(df, aes(sp)) + geom_bar(fill = got_col) +
       xlab("shortest path length") + theme_bw() +
       ggtitle("shortest path length distribution - GOT")
     print(p)
     ```



   - Using the function `as.dist` and `hclust`, transform the shortest path lengths into a `dist` object and use it to perform a hierachical clustering of GOT characters.



- Use the previous result to obtain 8 clusters and display the GOT graph with nodes colored with their cluster



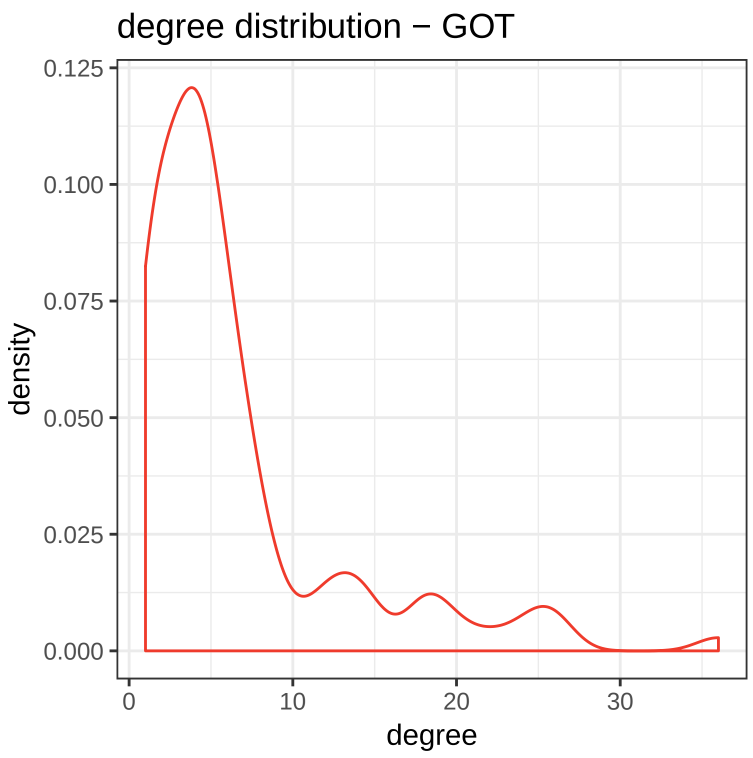
membership.

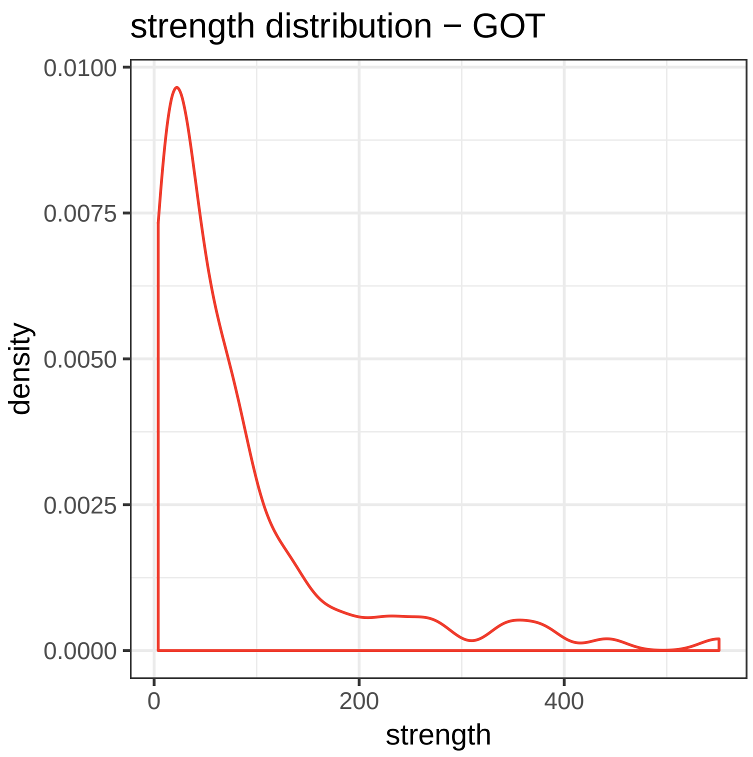
Exercice 3 Local characteristics

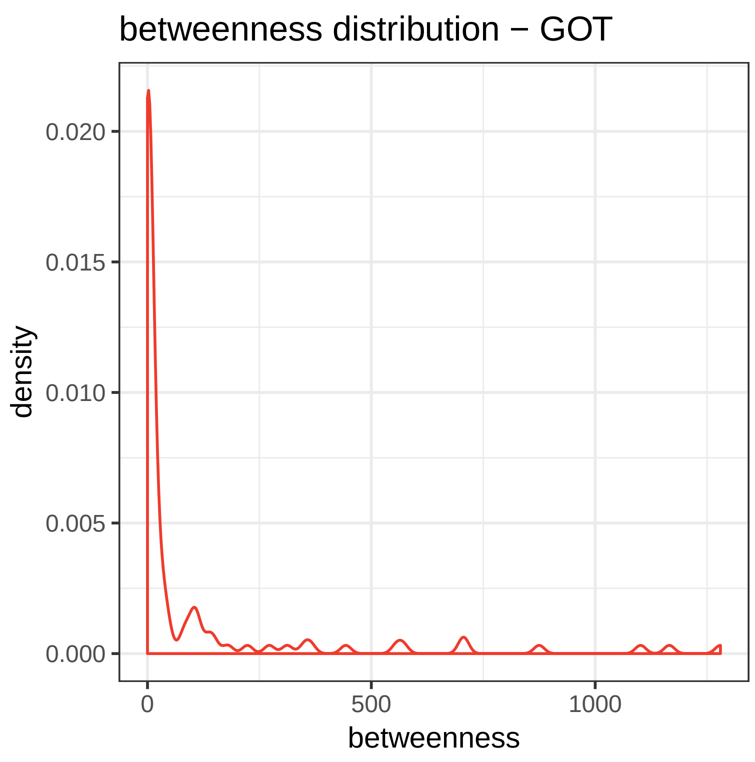
1. The degree, strength, betweenness, eccentricity and closeness of nodes are obtained with the functions `degree`, `strength`, `betweenness`, `eccentricity` and `closeness`. Build a data frame with this characteristic calculated for all vertices in GOT network and print a summary. Name this data frame `got_char` with column names `degree`, `strength`, `betweenness`, `eccentricity` and `closeness`. Be careful to use the unweighted version of the betweenness.

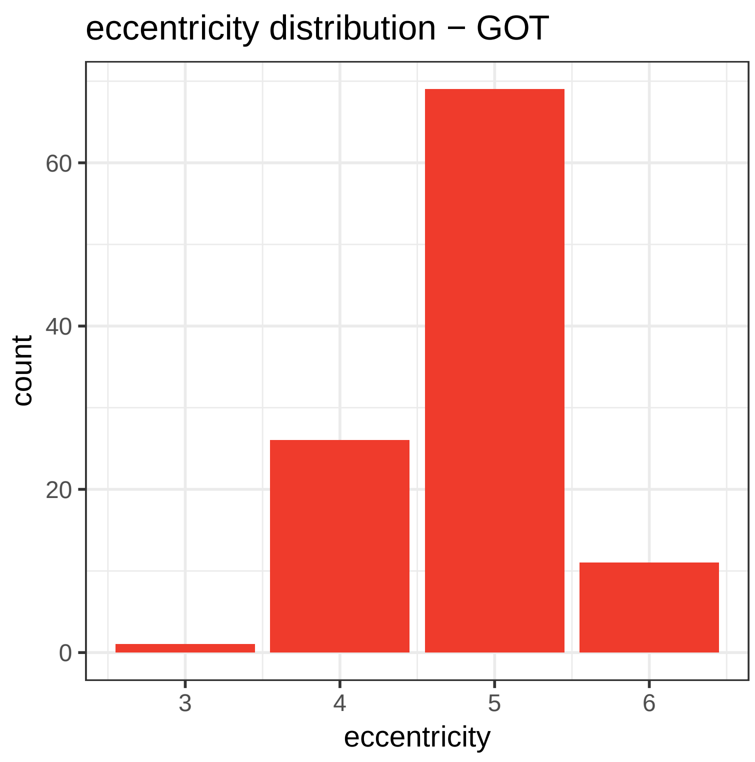
```
##      degree          strength       betweenness       eccentricity
##  Min.   : 1.000   Min.   :  4.00   Min.   :   0.00   Min.   :3.000
##  1st Qu.: 2.000   1st Qu.: 18.50   1st Qu.:   0.00   1st Qu.:4.500
##  Median : 4.000   Median : 44.00   Median :   3.39   Median :5.000
##  Mean   : 6.579   Mean   : 80.82   Mean   : 100.91   Mean   :4.841
##  3rd Qu.: 7.000   3rd Qu.: 89.00   3rd Qu.:  44.30   3rd Qu.:5.000
##  Max.   :36.000   Max.   :551.00   Max.   :1279.75   Max.   :6.000
##    closeness
##  Min.   :0.0001588
##  1st Qu.:0.0003824
##  Median :0.0004695
##  Mean   :0.0004591
##  3rd Qu.:0.0005321
##  Max.   :0.0006605
```

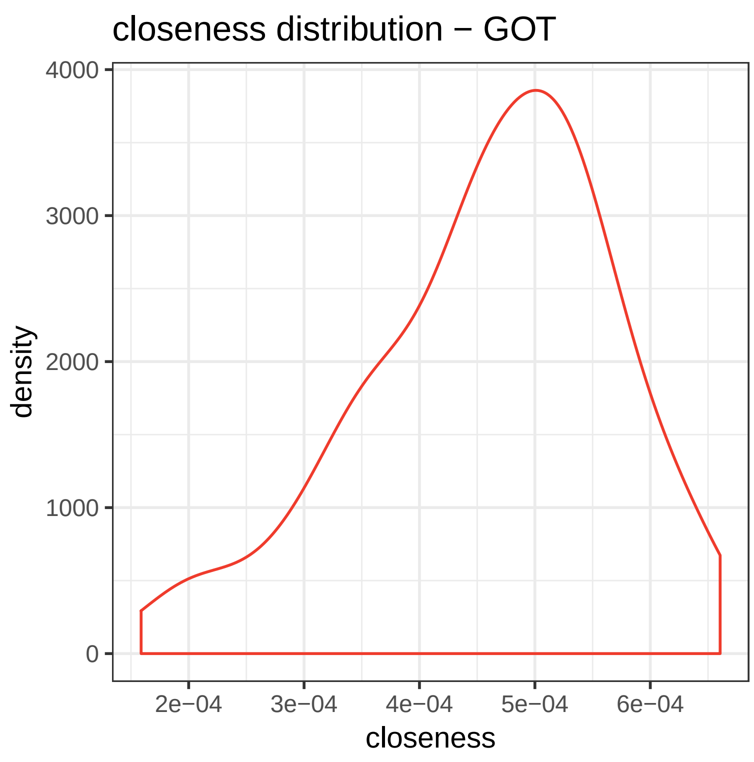
2. Print the distribution of these numerical characteristics (with a density plot for the degree, the strength, the betweenness and the closeness and with a barplot for the eccentricity).





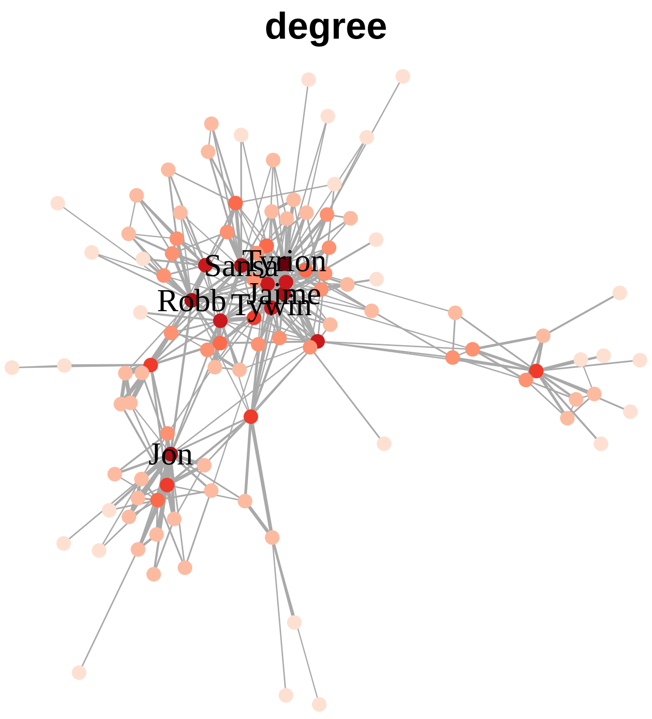

betweenness distribution − GOT
density
0.020
0.015
0.010
0.005
0.000
0
500
1000
betweenness


eccentricity distribution − GOT
count
60
40
20
0
3
4
5
6
eccentricity

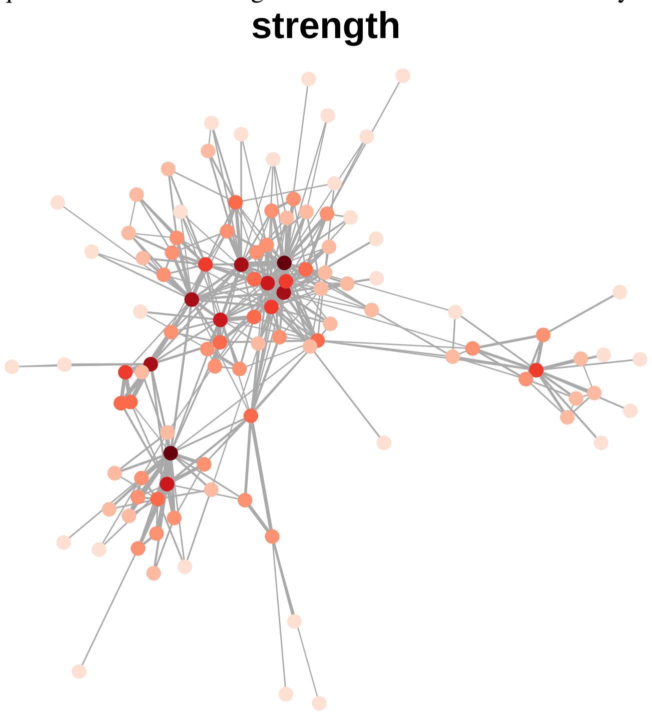


3. Display these information with a color code according to the characteristic value. To do so, use the function `cut` to split the values into 8 clusters with same width and print only the names of the vertices with the highest values (degree larger than 20 for instance). Values can be previously transformed if they are skewed (square root transformation for the degree):

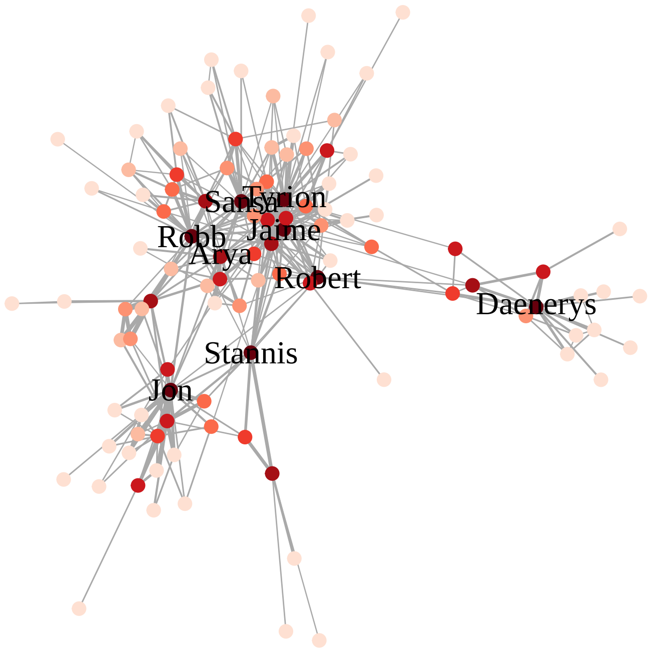
```
vcolor <- cut(sqrt(got_char$degree), breaks = 8, labels = FALSE)
vcolor <- brewer.pal(9, "Reds")[vcolor + 1]
vlabel <- V(got_net)$name
vlabel[got_char$degree <= 20] <- NA
par(mar = rep(1,4))
set.seed(1641)
plot(got_net, vertex.size = 4, vertex.frame.color = vcolor,
     vertex.color = vcolor, vertex.label.color = "black",
     edge.width = sqrt(E(got_net)$weight) / sqrt(max(E(got_net)$weight)) * 3,
     vertex.label = vlabel, main = "degree")
```



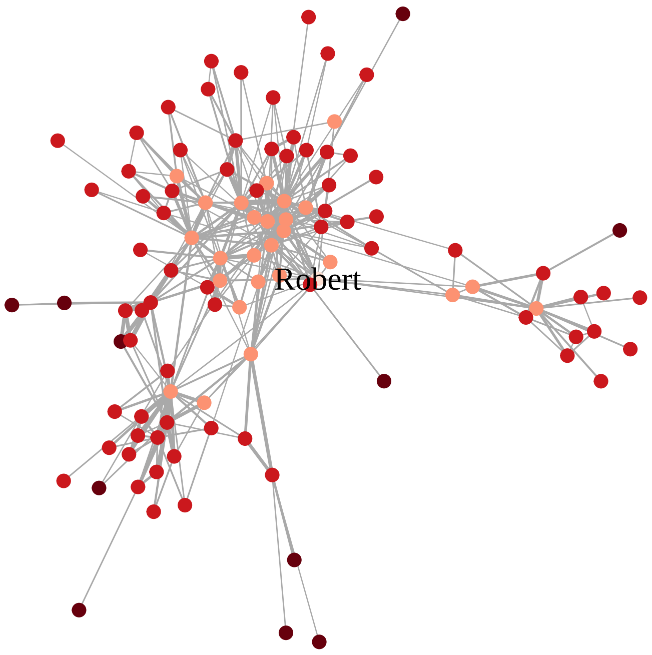
Strength, betweenness, eccentricity and closeness have respective thresholds (for vertex names) equal to 1,000, 400, 4 and 0.0006. The classes are obtained by transforming the original values with (respectively) the square root and the logarithm for the two first and by keeping the original values for the two last.



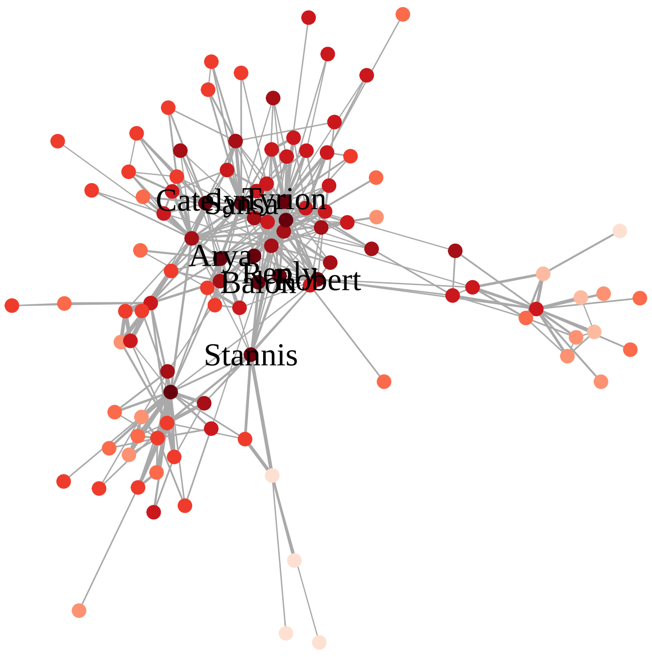
betweenness



eccentricity



closeness


Catelyn
Jon
Sansa
Tyrion
Arya
Renly
Balon
Robert
Stannis