Nathalie Vialaneix
Année 2019/2020

M2 in Statistics & Econometrics
Graph mining
Lesson 3 - Tests and random graphs

This worksheet illustrates the use of the R package **igraph** to generate random networks and to perform tests of significance on graphs. The packages **RColorBrewer**, **ggplot2** and **doMC** (**doParallel** for Windows users) will also be used in this worksheet. Start loading all the packages with:

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

The data used to illustrate this work can be found at http://www.nathalievialaneix.eu/doc/zip/data_M2SE.zip (for GOF 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 `lesson2.R` in another subdirectory (located in the same place than `data`) called `RLib`.

Exercice 1 Comparison with random graphs

This exercise uses the GOT network. Start the exercise by creating `got_net` as was done in worksheet 1.

```
## IGRAPH 6606f3d UNW- 107 352 --
## + attr: layout (g/n), name (v/c), weight (e/n)
## + edges from 6606f3d (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
```

1. The function `sample_gnm` is used to generate random graphs from the model $G(n,m)$. What does the following code perform?

```
set.seed(20041721)
B <- 100
global_char <- matrix(NA, nrow = B, ncol = 3)
for (ind in 1:B) {
  rg <- sample_gnm(vcount(got_net), ecount(got_net))
  if (is.connected(rg)) {
    global_char[ind, ] <- c(graph.density(rg), transitivity(rg, weights = NA),
                            diameter(rg, weights = NA))
```

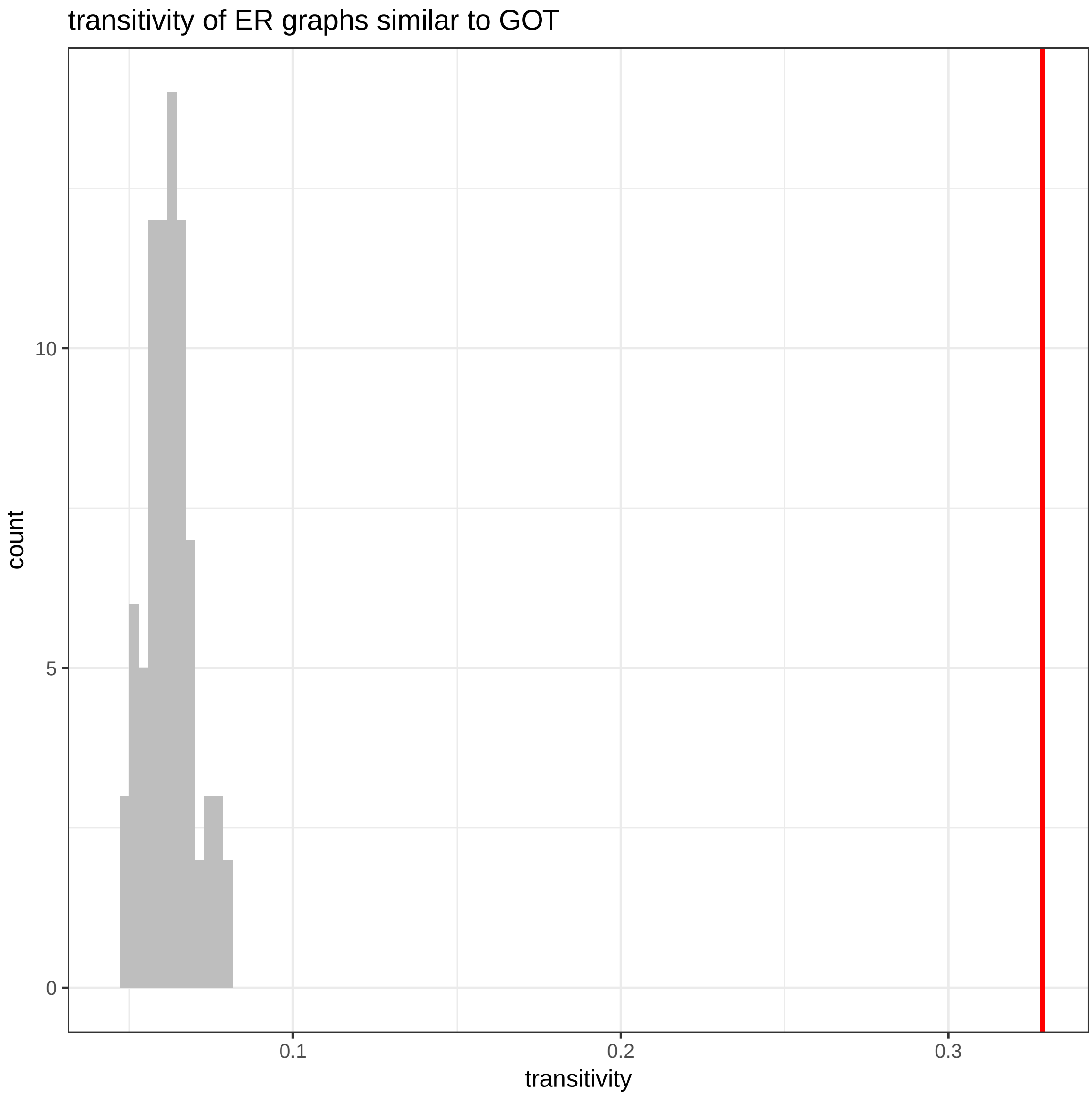
```
  }
}
colnames(global_char) <- c("density", "transitivity", "diameter")
global_char <- na.omit(global_char)
```

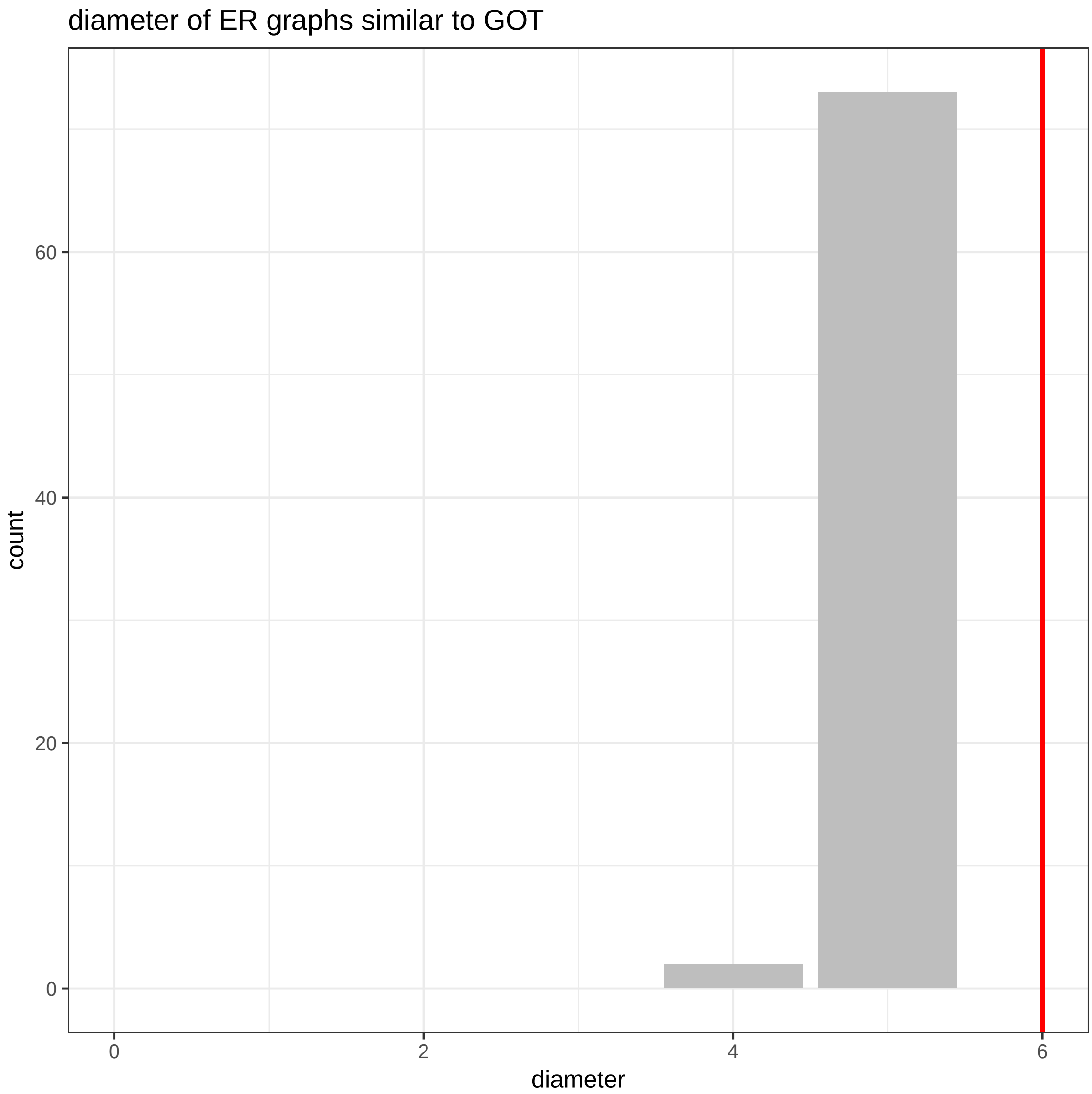
What does the ‘for‘ loop returns for a row that comes from a non connected graph? How is this problem handled? Using this script, answer to the following questions:

(a) How many generated graphs were connected?

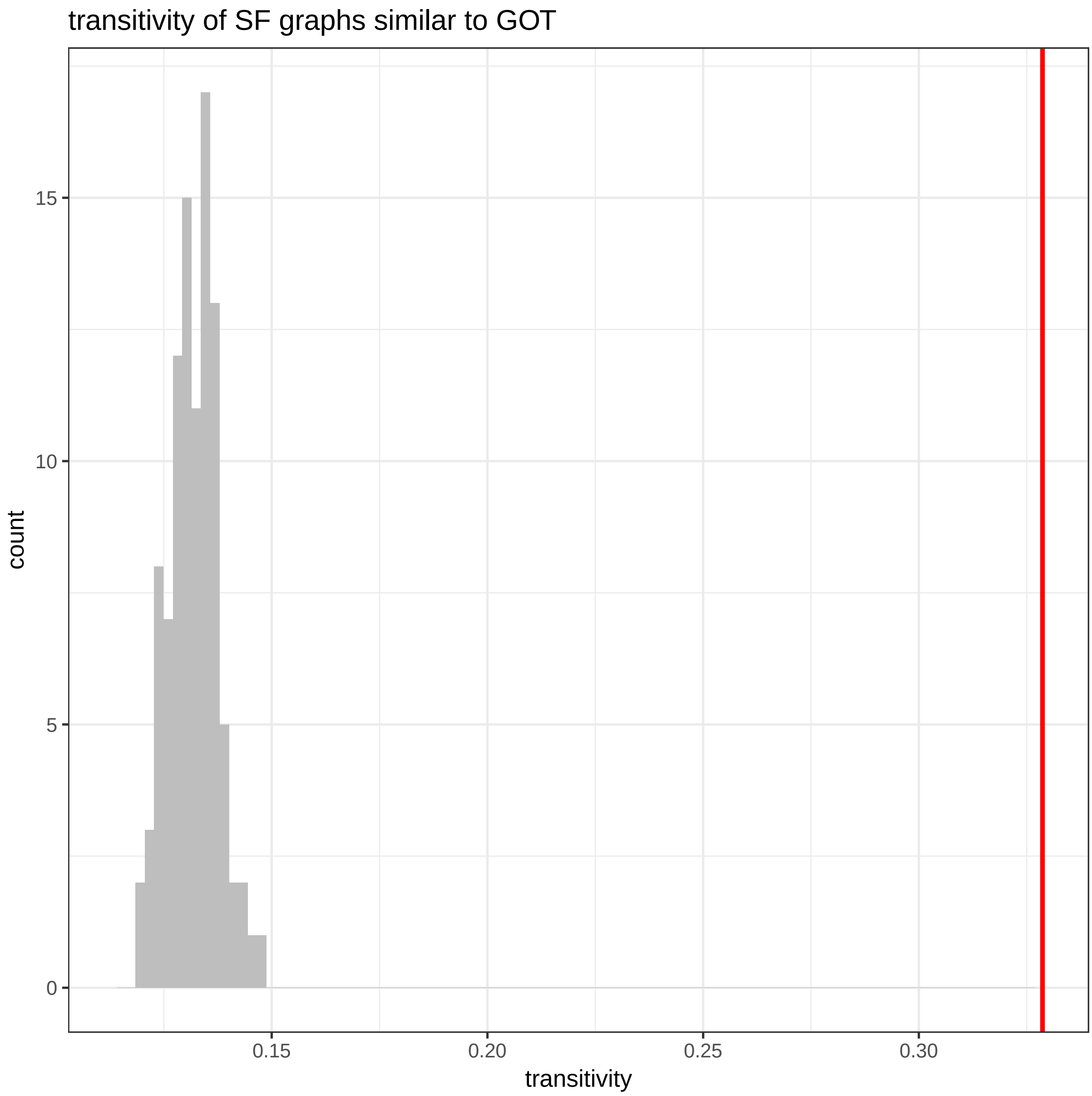
```
## [1] 82
```

(b) How do the transitivity and the diameter of these graphs compare to the transitivity of the real graph `got_net`?

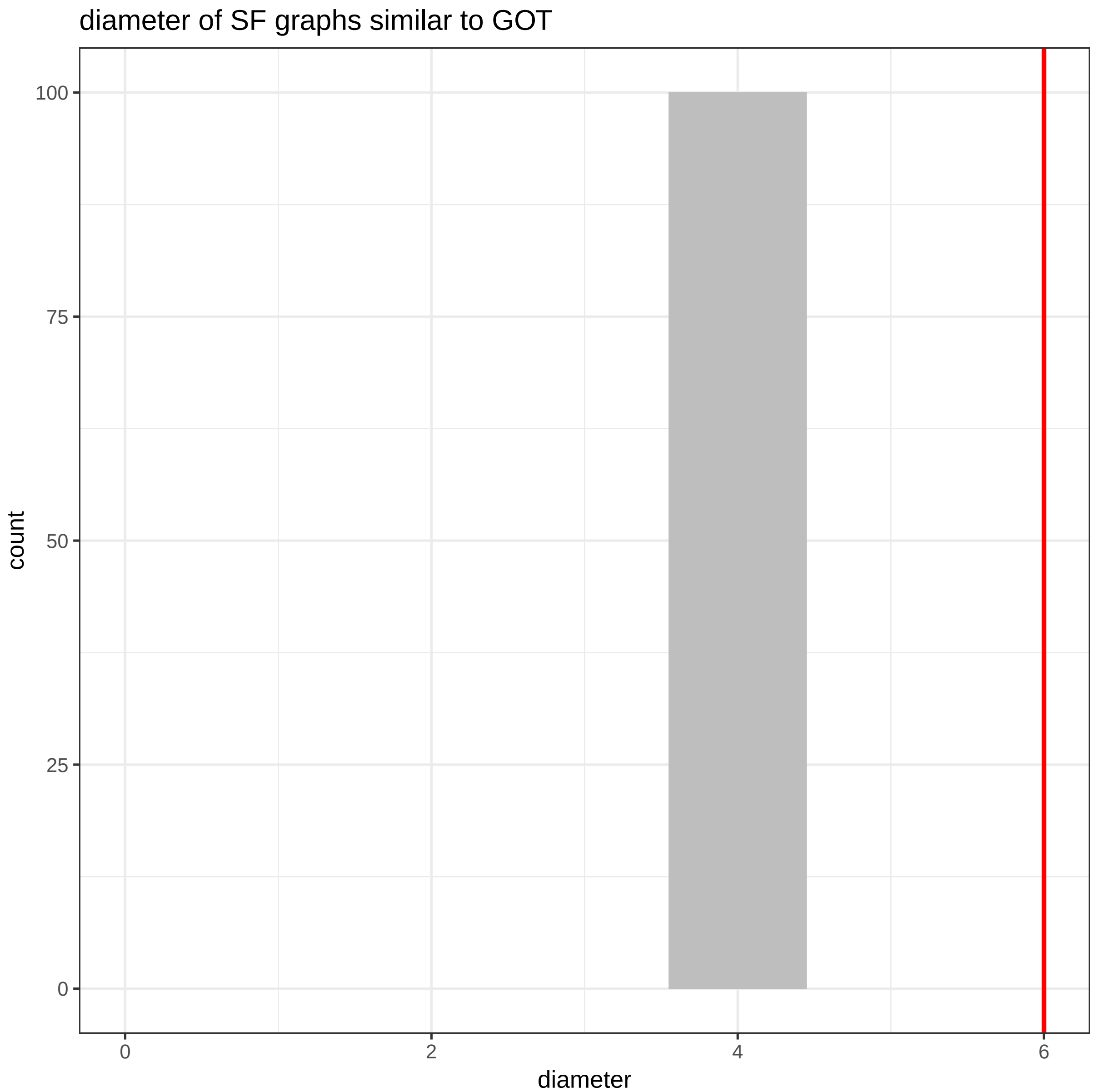




2. Use the same approach and answer the same question with the function `sample_pa` that generates scale free graphs according to Barabasi-Albert model. Use `m = 4` to run this function (how to tune this number is out of the scope of this lesson).



transitivity of SF graphs similar to GOT
count
15
10
5
0
0.15
0.20
0.25
0.30
transitivity



Exercice 2 Permutation tests

1. The function `rewire` is used to generate random graphs by randomly permuting two edge endpoints. The second argument `with` of this function specifies a function call to one of the rewiring method, `keeping_degseq` indicating to keep the degree distribution. What does the following code perform and which value to use for `Q`?
Be careful, when using this script, that it uses a parallel backend. For Windows, the proper parallel backend is handled with the functions of the R *package* `doParallel`.

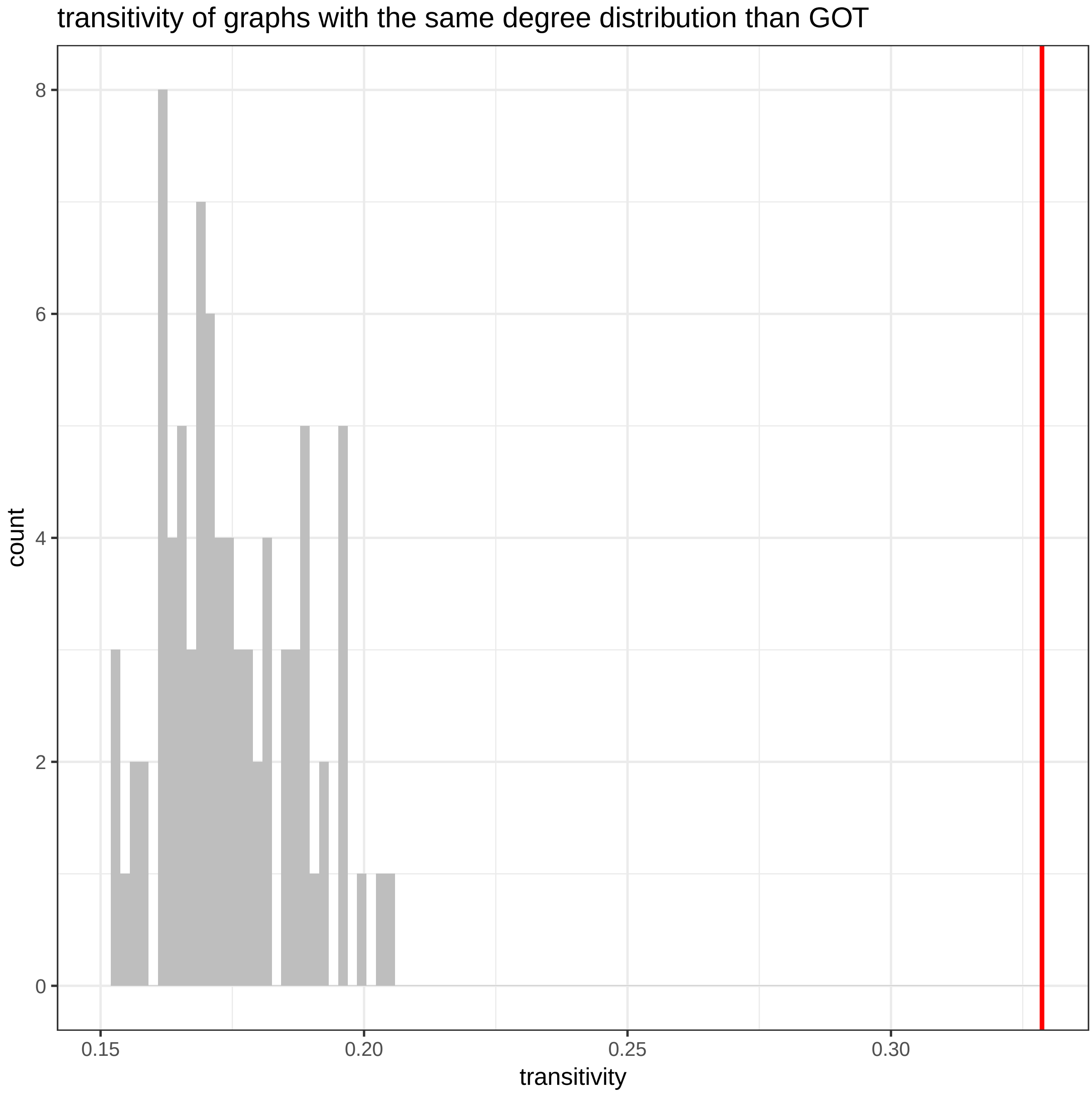
```
set.seed(22011600)
iter <- 100
B <- 100
all_seeds <- sample(1:22011600, B, replace = FALSE)
registerDoMC(cores = 7)
global_char <- foreach (ind=1:B, .combine = rbind) %dopar% {
  set.seed(all_seeds[ind])
  rg <- rewire(got_net, keeping_degseq(n = iter * Q))
  if (is.connected(rg) & is.simple(rg)) {
    res <- c("transitivity" = transitivity(rg, weights = NA),
             "diameter" = diameter(rg, weights = NA))
```

```
  } else res <- rep(NA, 2)
  return(res)
}
global_char <- na.omit(global_char)
```

How many of these networks were connected?

```
## [1] 84
```

2. Use the previous result to compare the transitivity of the observed graph with the transitivity of random graphs with the same degree distribution.



3. Use the same type of script to generate a list of `iter` betweenness distributions for random graphs with the same degree distribution than GOT. How many of these networks were connected?

```
set.seed(22011706)
iter <- 100
B <- 100
all_seeds <- sample(1:22011706, B, replace = FALSE)
registerDoMC(cores = 7)
```

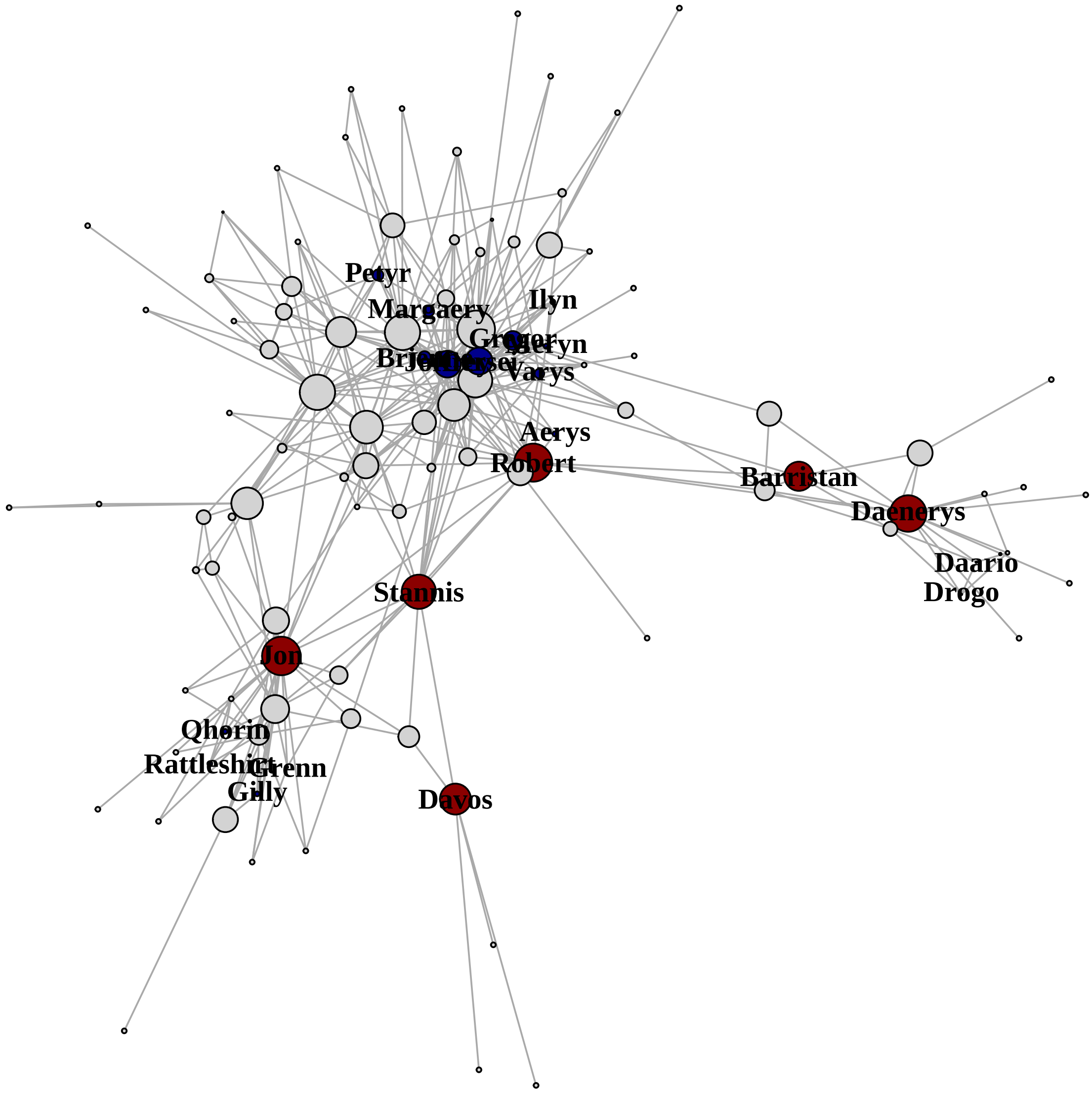
```r
global_char <- foreach (ind=1:B, .combine = rbind) %dopar% {
  set.seed(all_seeds[ind])
  rg <- rewire(got_net, keeping_degseq(n = iter * Q))
  if (is.connected(rg) & is.simple(rg)) {
    res <- betweenness(rg, weights = NA)
  } else res <- rep(NA, vcount(got_net))
  return(res)
}
global_char <- na.omit(global_char)
nrow(global_char)

## [1] 88
```

4. What does the following code compute?

```r
# obtain estimated p-values
bet_got <- betweenness(got_net, weights = NA)
valid_exp <- nrow(global_char)
c_betweenness <- rbind(bet_got, global_char)
p_high <- apply(c_betweenness, 2, function(acol)
  sum(acol[1] > acol[-1]) / valid_exp)
p_low <- apply(c_betweenness, 2, function(acol)
  sum(acol[1] < acol[-1]) / valid_exp)
```

5. Use the obtained `p_high` and `p_low` to obtain the following plot: blue nodes are those that have a betweenness lower than expected by random chance at risk 5%, red nodes are those with a betweenness larger than expected. Size of the nodes are proportionnal to the log-transformed betweenness and only blue and red nodes have their names displayed.


Petyr
Margaery
Ilyn
Varys
Aerys
Robert
Barristan
Daenerys
Daario
Drogo
Stannis
Jon
Qhorin
Grenn
Gilly
Davos