Entrega: Tarea 2

Laura Basalo Tur, Camila Pérez Arévalo, Josep Roman Cardell

We shall consider again the undirected Facebook friendship network considered in the last handout. The links in this network are contained in the file **facebook_sample_anon.txt**. Download it on your computer and upload it to R as a dataframe. Define an undirected graph with this list of edges.

```
#Cargamos los datos del fichero .txt en un dataframe
dataframe = read.table("data/facebook_sample_anon.txt",
                       header = FALSE,
                       col.names = c("nodeA", "nodeB"),
                       sep = " ")

#Generamos el grafo no dirigido desde los datos del dataframe
undirected_graph = graph_from_data_frame(dataframe, directed=F)
```

1) It has been observed in many networks an association between "centrality" and "lethality," defined as the fatal disconnection of the network when nodes are removed. Let's study this association on this network.

a) Repeat 1000 times the procedure of removing a random 0.1% of its set of nodes, and compute the average number of connected components of the resulting networks and the average fraction of the network represented by the largest component. Use **set.seed** to make your results reproducible.

```
#Indicamos la semilla a utilizar para que no cambien los resultados:
set.seed(4321)

#Número de repeticiones
n = 1000

compute_graph_random = function(g, n, perc){

  #Inicializamos dos listas vacías donde se guardarán los datos a calcular
  avrg_connected = c()
  avrg_largest_component = c()
  #Calculamos el porcentaje de nodos a eliminar
  perc = (0.1/100)*vcount(g)

  for(i in 1:n+1){
    #Obtenemos el 0.1% de vértices aleatorios a eliminar
    v_to_delete = sample(V(g), perc)
    g = delete_vertices(g, v_to_delete)
    #Obtenemos información de los componentes del grafo
    components = components(g)
    #Guardamos el número de componentes conexos
    avrg_connected = c(avrg_connected, components$no)
```

```
    #Guardamos la fracción del componente conexo más grande
    avrg_largest_component = c(avrg_largest_component, max(components$csize)/vcount(g))
  }

  result = list("Average number of connected components" = mean(avrg_connected),
                "Average fraction by largest component" = mean(avrg_largest_component))
  return(result)
}

#Ejecutamos la función del cálculo con el grafo no dirigido
compute_graph_random(undirected_graph, n, perc)
```

```
## $`Average number of connected components`
## [1] 50.245
##
## $`Average fraction by largest component`
## [1] 0.8232974
```

COMENTARIOS

b) Now, compute the number of connected components and the fraction represented by the largest component of the networks obtained after removing the most central 0.1% of nodes, for the following centrality indices (of course, if the most central 0.1% of nodes for two indices are the same set of nodes, you need not waste your time considering twice the same network): *degree*; *closeness*; *betweenness*; *page.rank*. (**Hint**: It might be convenient to define first a function that removes a given set of nodes of this graph and computes the number of connected components and the fraction represented by the largest component of the resulting network; then you will only need to apply it to the required different sets of most central nodes.) Is it what you expected?

```
order_by_degree = order(centr_degree(undirected_graph)$res, decreasing = TRUE)
v_to_delete = order_by_degree[1:perc]
```

```
## $`Average number of connected components`
## [1] 375.779
##
## $`Average fraction by largest component`
## [1] 0.3619654
```

```
order_by_clos = order(centr_clo(undirected_graph)$res, decreasing = TRUE)
v_to_delete = order_by_clos[1:perc]
```

```
order_by_betwennes = order(centr_betw(undirected_graph)$res, decreasing = TRUE)
v_to_delete = order_by_betwennes[1:perc]
```

```
## $`Average number of connected components`
## [1] 405.021
##
## $`Average fraction by largest component`
## [1] 0.06844139
```

```
page_rank_result = page_rank(undirected_graph, directed = FALSE, algo = "power")
order_by_pagerank = order(page_rank_result$vector, decreasing = TRUE)
v_to_delete = order_by_pagerank[1:perc]
```

```
## $`Average number of connected components`
## [1] 420.446
##
## $`Average fraction by largest component`
## [1] 0.2768924
```

Sí, esperabamos que la fracción del componente más largo, en el apartado b, fuese menor que en el anterior apartado ya que, en este caso se eliminan aquellos vértices que cuentan con un mayor grado. Es decir, se eliminan los vértices que cuentan con el mayor nombre de aristas de todo el grafo, por lo que sería más probable que el grafo se divida en más componentes y, como resultado, el tamaño del componente más largo se vea disminuido.

A la hora de realizar el closeness nos aparecia el siguiente error: ***At centrality.c:2784 :closeness centrality is not well-defined for disconnected graphs***, lo que sucede es que esta función da error una vez existe la división de grafos. Por este motivo hemos aplicado una restricción en la cual por cada iteración elimine 4 vértices de el componente más largo. De esta manera se ha podido ejecutar el código sin que nos apareciese este error pero llega un momento en el cual los componentes son de tamaño 2 y 1 por lo que closeness deja de ejecutarse.

2) Now, consider the same graph as a directed one, and find the hubs and authorities scores. Compare with the page rank score.

```
#Generamos el grafo dirigido desde los datos del dataframe
directed_graph = graph_from_data_frame(dataframe, directed=T)

#Hub/Authorities
hub_score = hub_score(directed_graph)
authority_score = authority_score(directed_graph)

#PageRank
page_rank(directed_graph, directed = TRUE, algo = "power")
```