Entrega: Tarea 1

Laura Basalo Tur, Camila Pérez Arévalo

7/11/2020

Ejercicio 1

The file *facebook_sample_anon.txt* is a data table containing the list of edges of an anonymized sample of the Facebook friendship network. Download it on your computer, upload it to R as a dataframe, and 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)
```

a) Is it connected? If it is not, replace it by its largest connected component.

Un grafo es conexo si cada par de vértices está conectado por un camino; es decir, si para cualquier par de vértices (a, b), existe al menos un camino posible desde *'a'* hacia *'b'*.

Para comprobar si el grafo guardado en la variable *undirected_graph* es conexo se ha utilizado la función *is.connected*.

```
is_connect = is.connected(undirected_graph)
```

El resultado obtenido es TRUE, por lo que podemos afirmar que el grafo es conexo.

b) Compute the edge density.

La densidad de un grafo es la relación entre el número de aristas del grafo y el número de aristas máximo. Para calcular la densidad se ha utilizado la función *edge_density*, indicando que no tenga en cuenta ningún posible loop del grafo.

```
density = edge_density(undirected_graph, loops=F)
```

El resultado obtenido es 0.01082, por lo que podemos deducir que el grafo es disperso.

c) What is the mean distance among the subjects?

La distancia entre dos vértices de un grafo es el número de vértices mínimo que debe recorrerse para unirlos. Para calcular la media de esta distancia se ha utilizado la función *mean_distance*.

```
mean_distance = mean_distance(undirected_graph, directed=F)
```

El resultado obtenido de la distancia media es 3.6925068.

d) Calculate the list of vertices in a diameter of the graph. Plot only this path with the size of the node proportional to the degree of the node.

El diámetro de un grafo es la mayor distancia entre todos los pares de puntos de la misma. A continuación se obtiene el listado de vértices del diametro del grafo y se representa un gráfico con el camino del mismo.

```
#Calculamos y guardamos el grado de cada vértice del grafo
V(undirected_graph)$v_dg = degree(undirected_graph)

#Obtenemos el diametro del grafo
diameter = get_diameter(undirected_graph, directed=F)

#Creamos el subgrafo a partir de los vértices que forman el diámetro del grafo original
subgraph = induced_subgraph(undirected_graph, V(undirected_graph)[diameter])

#Indicamos el layaout a utilizar
set.seed(1234)
lo = layout.fruchterman.reingold(subgraph)
lo <- norm_coords(lo, ymin=-1, ymax=1, xmin=-1, xmax=1)

#Asociamos un color a cada vértice en función del grado del vértice
V(subgraph)[V(subgraph)$v_dg < 50]$color = "khaki1"
V(subgraph)[V(subgraph)$v_dg > 50 & V(subgraph)$v_dg < 100]$color = "goldenrod1"
V(subgraph)[V(subgraph)$v_dg > 100 & V(subgraph)$v_dg < 300]$color = "lightcoral"
V(subgraph)[V(subgraph)$v_dg > 300]$color = "tomato2"

#Generamos el plot del subgrafo
plot(subgraph,
     main = "Diámetro del grafo y grado de los vértices",
     layout = lo,
     vertex.frame.color = "black",
     vertex.size = V(subgraph)$v_dg/10 + 3,
     vertex.label.color= "black",
     vertex.label.font= 2,
     vertex.label.cex= 1,
     vertex.label.dist= 5.3,
     edge.color= "gray80",
     edge.width= 2
     )
```

El resultado y representación del subgrafo obtenido del diámetro del grafo principal se puede observar en la Figura 1. El diámetro del grafo es el siguiente listado de vértices: 585, 584, 595, 3156, 495, 352, 515, 3624, 3625 y en la representación el tamaño de su respectivo nodo es proporcional a su grado.

e) Calculate the shortest path from the vertex named "1000" to the vertex named "2000" in the original file.

Para calcular el camino más corto entre los vértices se ha utilizado la función *shortest_paths*.

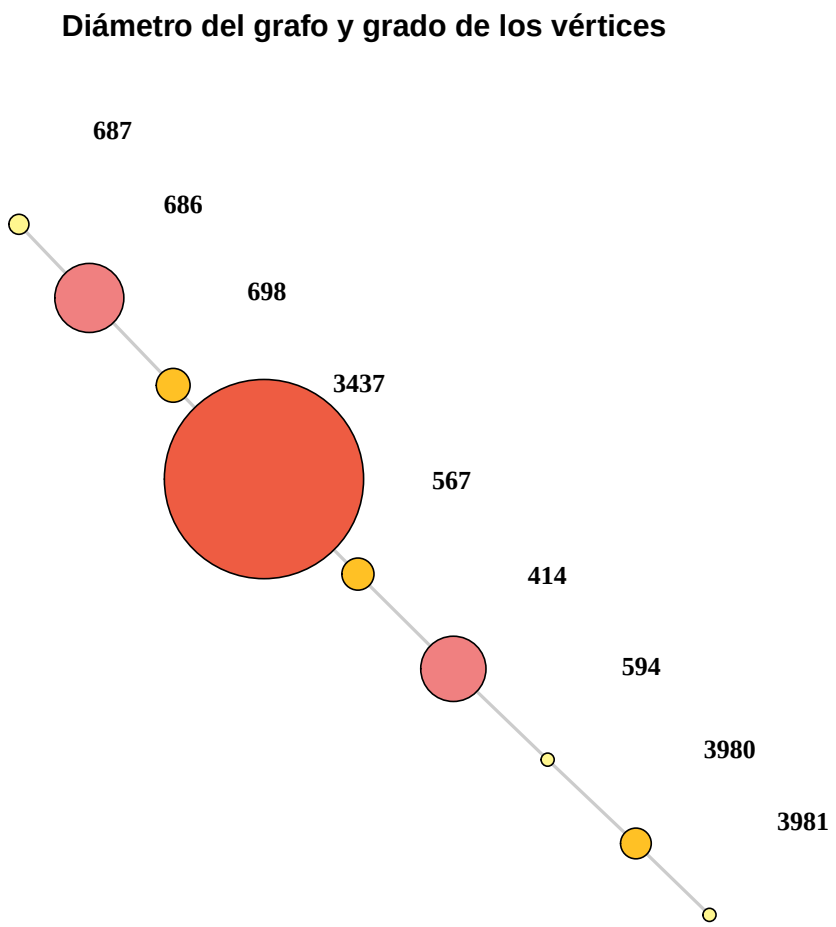

Figure 1: Representación gráfica del diámetro del grafo principal.

```r
shortest = shortest_paths(undirected_graph,
                          from = V(undirected_graph)["1000"],
                          to  = V(undirected_graph)["2000"],
                          output = "both") # both path nodes and edges
```

Se observa que el resultado del camino más corto, en vértices, es el siguiente: c(1000 = 868, 107 = 100, 58 = 52, 1912 = 1723, 2000 = 1810). r

f) Calculate a clique of 5 friends, if there is one.

Un clique es un subgrafo en el que cada vértice está conectado a todos los demás vértices del subgrafo. Para calcular los cliques de un grafo se puede utilizar la función *cliques*. En este caso, al ser un grafo de gran magnitud, la complejidad para encontrar los cliques es muy alta y tardaría horas en obtener un resultado.

De forma teórica se realizaría el cálculo de la siguiente manera:

```r
#cliques(undirected_graph, min = 5, max = 5)
```

h) Calculate the list of names of verteces that are the neigbours of verteces of degree one and that are not of degree one.

Para calcular el listado vértices de grado diferente a 1 y con vecinos con grado igual a 1 realizamos los siguientes pasos:

```r
#Calculamos el grado de todos los vértices y guardamos su resultado
V(undirected_graph)$degree = degree(undirected_graph, mode="all")

#Filtramos los vértices cuyo grado es 1 (solo tienen 1 vecino)
vertices_1_grado = V(undirected_graph)[degree(undirected_graph)==1]

#Obtenemos la lista de vértices que son adyacentes a los vértices de grado 1
vecinos = adjacent_vertices(undirected_graph, v = vertices_1_grado)
vecinos = unlist(vecinos)

#Obtenemos el listado de vértices a partir de sus posiciones
nombre_vecinos = V(undirected_graph)[vecinos]

#Eliminamos vértices duplicados
resultado = unique(nombre_vecinos)
```

La lista de vértices es: 1, 100, 288, 352, 584, 595, 1535, 1723, 3156, 3624.

i) Extra: Programa que busque una clique de X tamaño y pare al encontrarla.

El problema de encontrar cliques en un grafo se resuelve mediante el algoritmo de Bron–Kerbosch, que en pseudocódigo se puede representar de la siguiente manera:

```
algorithm BronKerbosch1(R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    for each vertex v in P do
        BronKerbosch1(R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

Donde R y X son vectores vacíos y P los vértices del grafo del cual se quieren hallar los cliques. Pasando este código a `R`, se observa como:

```
bron_kerbosch <- function(R,P,X) {

    if (length(P) == 0 && length(X) == 0) {
      print (R)
    } else {
        for (v in P) {
            neis <- neighbors(g,v)$name
            bron_kerbosch(union(R,v),intersect(P,neis),intersect(X,neis))
            P <- c(P[-which(P==v)])
            X <- c(X,v)
        }
    }
}
```

Con este algoritmo se imprimirían todos los cliques del grafo, hasta el clique máximo. Para controlar en qué clique parar de calcular se debería añadir una condición en el *if* que imprimiese solamente cuando la longitud del parámetro R fuese el número del tamaño del clique. Por ejemplo, en el siguiente código de bloque se muestra el algoritmo con la nueva condición añadida para cliques de tamaño 4:

```
bronkerbosch <- function(R,P,X) {

    if (length(P) == 0 && length(X) == 0 && length(R) == 4) {
      print (R)
    } else {
        for (v in P) {
            neighbors <- neighbors(g,v)$name
            bronkerbosch(union(R,v),intersect(P,neighbors),intersect(X,neighbors))
            P <- c(P[-which(P==v)])
            X <- c(X,v)
        }
    }
}
```