Módulo 2 - Grafos

Jorge Fábrega

Año 2023

Introducción

La teoría de grafos, originada en el siglo XVIII con Leonhard Euler y su análisis del problema de los puentes de Königsberg, es una rama matemática e informática que estudia las propiedades y aplicaciones de grafos. Estos grafos, compuestos por vértices y aristas, modelan relaciones entre objetos y se han expandido en aplicabilidad a áreas como la informática, ingeniería, biología y el conjunto de las ciencias sociales. Por ejemplo, en informática, facilitan la representación de redes de comunicación, tales como enlaces web y rutas de transporte; y en ciencia política, se ha utilizado para revelar las relaciones informales que dan más poder a algunos actores respecto de otros.

En el análisis de redes sociales, la teoría de grafos es fundamental para visualizar y examinar interacciones complejas entre individuos y organizaciones. Permite identificar influenciadores y líderes de opinión, entender la difusión de información y detectar grupos dentro de redes más amplias. Además, provee herramientas para medir centralidad de nodos, densidad de redes y otras métricas clave en la comprensión de la dinámica social.

Esta disciplina es esencial en el análisis cuantitativo de redes sociales, no solo para estudiar su estructura sino también su evolución temporal. Al modelar relaciones sociales como grafos, se aplican algoritmos para identificar patrones y predecir cambios, incluyendo la optimización de algoritmos de recomendación y búsqueda en plataformas sociales.

Un uso destacado de la teoría de grafos es en el análisis de la difusión de noticias falsas en redes sociales digitales como Facebook o Twitter y cómo ello incide o no en la opinión pública o las elecciones. Representando cuentas de usuario como nodos y sus conexiones como aristas, se puede identificar la propagación de estas noticias y los grupos más susceptibles de ser influeciados por ellas, revelando cuentas clave en la difusión de información y su impacto en elecciones presidenciales (véase por ejemplo Knuutila et al 2022).

Breve Historia del Desarrollo de la Teoría de Grafos

La teoría de grafos comenzó con Leonhard Euler y el problema de los puentes de Königsberg en 1736. Este problema planteaba la cuestión de si era posible cruzar todos los puentes de la ciudad de Königsberg sin pasar por el mismo puente más de una vez. Euler abordó este problema utilizando una estructura que se asemeja a los grafos modernos, representando los puentes y las masas de tierra como aristas y nodos, respectivamente. Su análisis y solución de este problema marcaron el nacimiento de la teoría de grafos como un campo de estudio matemático, sentando las bases para futuros desarrollos en la materia.

Durante el siglo XX, la teoría de grafos experimentó un desarrollo significativo, en gran parte impulsado por el matemático húngaro Paul Erdős. Antes de Erdős, esta rama de las matemáticas ya estaba establecida. Sin embargo, Erdős fue un catalizador clave en el avance de la teoría de grafos, gracias a su extensa producción de investigación y colaboraciones con matemáticos de todo el mundo. Sus colaboraciones dieron lugar a nuevos conceptos y teoremas en la teoría de grafos, marcando un impacto duradero en el campo. Uno de los avances de más profundo impacto del trabajo de Erdös fue la teoría de grafos aleatorios cuyas implicancias la veremos en el módulo 6.

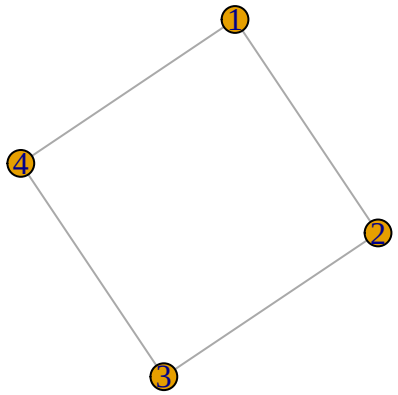
Después de Erdős, su influencia en la teoría de grafos y las matemáticas discretas continuó a través del trabajo de sus colaboradores y la comunidad matemática en general. Otra área central ha sido el desarrollo de

algoritmos que, basados en la estructura de una red, generan métodos de optimización, clasificación, etcétera para revelar diversas características de un grafo.

Representación Matemática de un Grafo

Consideremos un grafo simple $G = (V, E)$ con $V = \{1, 2, 3, 4\}$ y $E = \{\{1, 2\}, \{2, 3\}, \{3, 4\}, \{4, 1\}\}$. Este grafo puede simbolizar una red social con cuatro individuos, donde las aristas representan relaciones de amistad.

En términos matemáticos, este grafo puede representarse mediante una matriz de adyacencia o una lista de adyacencia. Por ejemplo, la matriz de adyacencia para este grafo sería una matriz 4×4 donde las entradas indican si hay una conexión entre los nodos. En este caso, las entradas diagonales serían cero, reflejando la ausencia de bucles (es decir, un individuo no se considera amigo de sí mismo), y las entradas (1,2), (2,3),(3,4) y (4,1) serían 1, indicando una relación de amistad. Alternativamente, en una lista de adyacencia, simplemente enumeraríamos para cada nodo los nodos con los que está conectado, proporcionando una representación más eficiente para grafos dispersos.

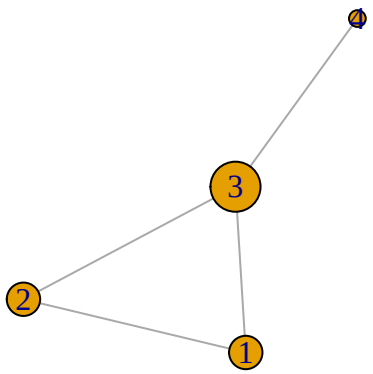


Grado de un Vértice (Vertex degree)

El grado de un vértice v, denotado como $deg(v)$, en un grafo $G = (V, E)$ es el número de aristas incidentes en v. Matemáticamente,

$$deg(v) = |\{e \in E : v \in e\}|$$

Otro convención común en los textos es que se represente con la letra k.

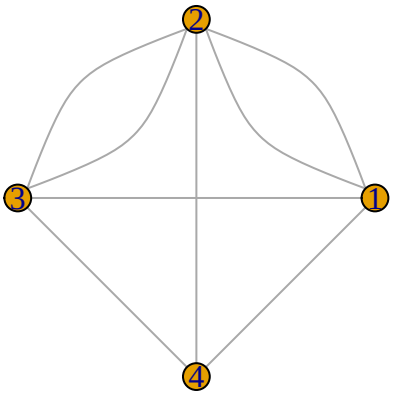


El problema de los puentes de Königsberg

Königsberg (ahora Kaliningrado, Rusia) estaba situada en ambos lados del río Pregel y estaba conectada por siete puentes. El desafío que se le presentaba a los visitantes era encontrar un paseo que cruzara cada puente exactamente una vez.

Leonhard Euler demostró en 1736 que no era posible encontrar tal paseo, y en el proceso sentó las bases para la teoría de grafos. Matemáticamente, Euler demostró que para que exista un camino que atraviese cada arista exactamente una vez (un recorrido de Euler), cada vértice del grafo debe tener un grado par (un número par de aristas incidentes), excepto para dos vértices (si el camino no es un ciclo) que pueden tener un grado impar.

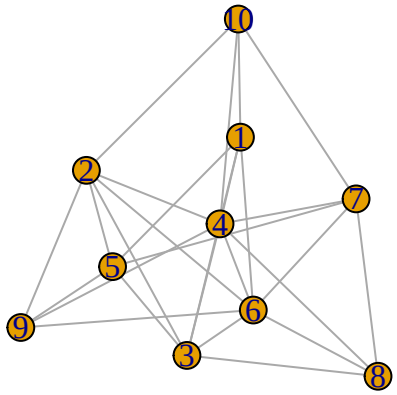
El grafo de Königsberg se representa con cuatro nodos, cada uno representando una de las cuatro masas de tierra, y siete aristas, cada una representando uno de los puentes.



El teorema de la mano

El Teorema de la mano, también conocido como el lema del apretón de manos, es un resultado fundamental en la teoría de grafos que afirma que en cualquier grafo (o red), la suma de los grados de todos los vértices es el doble del número de aristas. Esto se debe a que cada arista contribuye exactamente dos veces a la suma total de los grados: una por cada extremo.

```
set.seed(42)  # Para reproducibilidad
g <- erdos.renyi.game(n = 10, p = 0.5, type = "gnp", directed = FALSE)
plot(g)
```




```
degree_vector <- degree(g)
sum_degrees <- sum(degree_vector)
print("Los grados de cada nodo son:")
```

```
## [1] "Los grados de cada nodo son:"
```

```
print(degree_vector)
```

```
##  [1] 5 6 6 8 5 7 5 4 4 4
```

```
print(paste0("La suma de los grados es:",sum_degrees))
```

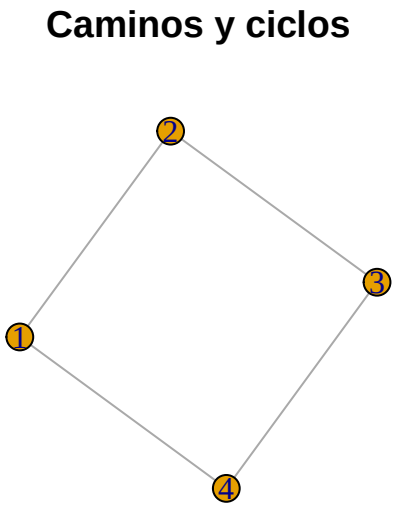
```
## [1] "La suma de los grados es:54"
```

```
num_edges <- ecount(g)
print(paste0("El número de links es:", num_edges))
```

```
## [1] "El número de links es:27"
```

Caminos y Ciclos (Paths and cycles)

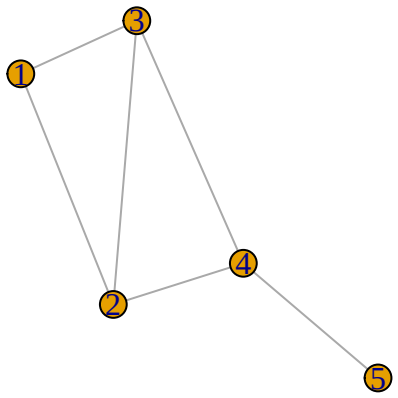
Un camino P en un grafo G es una secuencia de vértices $v_1, v_2, \ldots, v_k$ tal que $\{v_i, v_{i+1}\} \in E$ para todo $1 \leq i < k$. Un ciclo C es un camino donde el primer y último vértices son iguales, $v_1 = v_k$, y todos los demás vértices son distintos.



Conectividad (Connectivity)

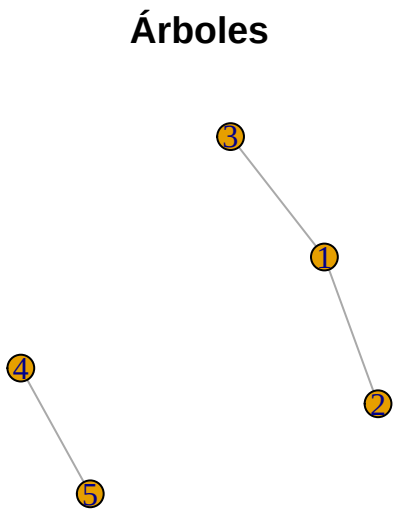
La conectividad $\kappa(G)$ de un grafo G es el mínimo número de vértices cuya eliminación resulta en un grafo desconectado o trivial. Matemáticamente,

$\kappa(G) = \min\{|S| : G - S \; es \; desconectado \; o \; trivial\}, donde \; S \subseteq V.$



Árboles y Bosques (Trees and forests)

Un árbol es un grafo $T = (V, E)$ que es conectado y acíclico. Un bosque es un grafo acíclico que puede estar desconectado, es decir, es una unión disjunta de árboles.



En un árbol con n nodos o vértices, siempre hay $n-1$ links o aristas. En un árbol, la eliminación de cualquier arista resulta en un grafo desconectado, mientras que la adición de cualquier arista crea un ciclo.

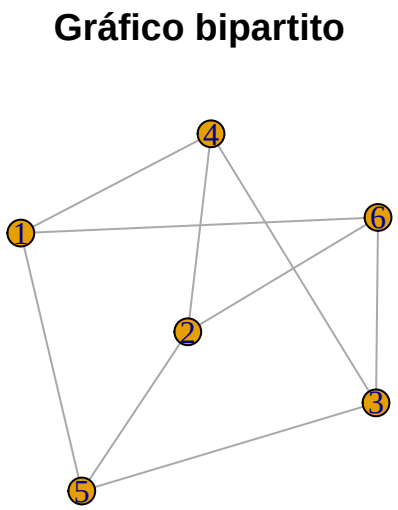
Los **árboles** son los mínimos grafos conexos; es decir, son grafos con el mínimo número de aristas para mantener la conectividad. - Los árboles sirven como estructuras fundamentales para entender conceptos más complejos en la teoría de grafos. Muchos problemas en grafos pueden ser simplificados o resueltos eficientemente cuando se consideran bajo la estructura de un árbol.
- Los árboles son importantes en problemas de optimización

Por su parte, un **bosque** puede ser visto como una generalización de un árbol; es útil en escenarios donde se tienen múltiples componentes desconectadas. Los bosques son importantes en el análisis de la conectividad de grafos y en algoritmos que operan en grafos desconectados.

Grafos Bipartitos (Bipartite graphs)

Los grafos bipartitos son una clase importante de grafos en la teoría de grafos y tienen aplicaciones significativas en diversos campos. Un grafo bipartito es aquel cuyo conjunto de vértices se puede dividir en dos conjuntos disjuntos, de manera que cada arista conecta un vértice de un conjunto con un vértice del otro conjunto, y no hay aristas entre vértices del mismo conjunto.

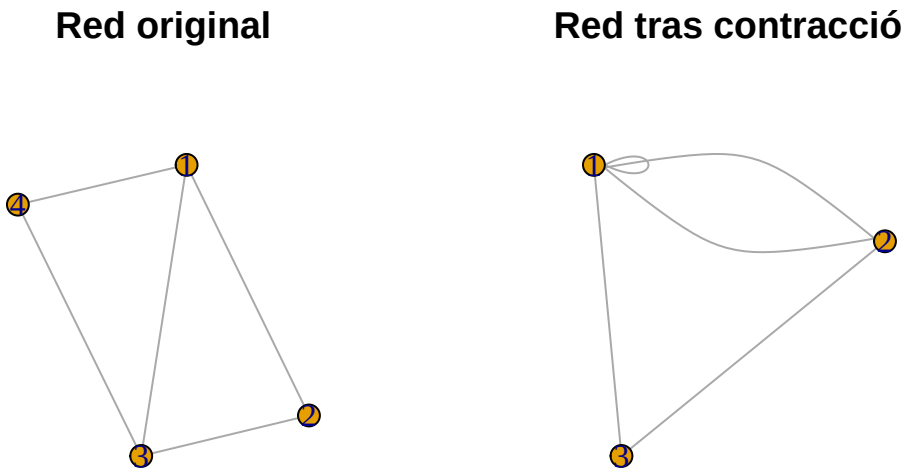
Por lo tanto, decimos que un grafo $G = (V, E)$ es bipartito si existe una partición de V en dos conjuntos disjuntos V_1 y V_2 tal que cada arista conecta un vértice en V_1 con un vértice en V_2. Esto se puede expresar como $E \subseteq V_1 \times V_2$. En el siguiente ejemplo hay tres nodos conectados con otros tres nodos, pero no entre sí.



Ejemplos de utilidad: - Sistemas de recomendación: Usuarios y películas en Netflix.
- Formación de conocimiento científico: Científicos y temáticas.
- Interlocking: Directores y empresas.

Contracción y Menores (Contraction and minors)

La contracción de una arista e en un grafo G es la operación que elimina e y fusiona los vértices que conecta. Un menor de un grafo G es un grafo que se puede obtener a partir de G mediante la eliminación de aristas y vértices y/o la contracción de aristas.

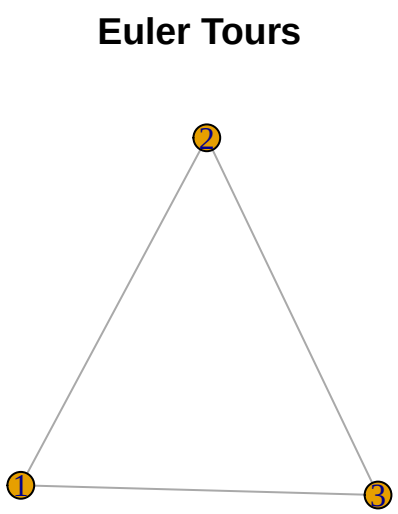


La contracción se utiliza para simplificar grafos, manteniendo ciertas propiedades estructurales. Es crucial en algoritmos que buscan resolver problemas como la identificación de cortes mínimos y en la teoría de gráficos topológicos.

Los menores son útiles para entender la resiliencia de la red frente a fallas o ataques, ya que permiten estudiar cómo se comporta la red al eliminar o contraer nodos y aristas.

Recorridos de Euler (Euler tours)

Un recorrido de Euler en un grafo G es un ciclo que utiliza cada arista exactamente una vez. Un grafo G tiene un recorrido de Euler si y solo si G es conexo y cada vértice tiene grado par, es decir, $extdeg(v)$ es par para todo $v \in V$ (Nota: extdeg = external o out degree, más sobre esto más adelante).

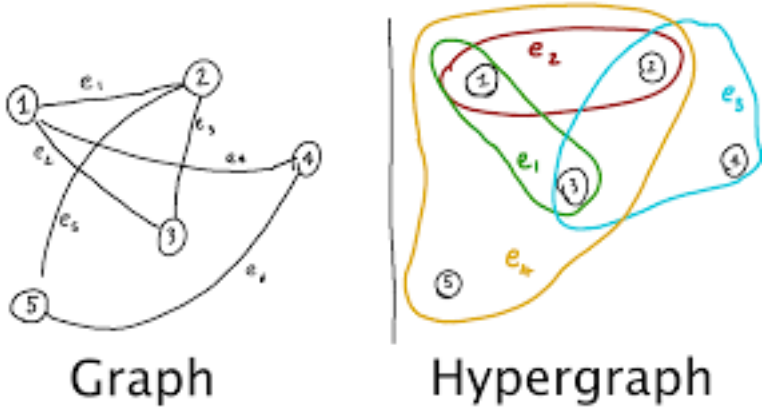


Los recorridos de Euler son útiles en problemas de optimización (por ejemplo, al repartir productos en una empresa de correos), como minimizar el recorrido en un sistema de rutas o caminos donde es necesario pasar por cada segmento exactamente una vez. Por ejemplo, en ocasiones una red no permite un recorrido de Euler (como en los puentes de Konigsberg), entonces, en esos casos hay que considerar la creación de nuevas aristas o nodos para poder optimizar.

Hipergrafo (Hypergraph)

Los hipergrafos permiten modelar relaciones entre conjuntos de elementos, en lugar de solo pares de elementos. Esto los hace especialmente útiles en situaciones donde las interacciones o conexiones no son simplemente binarias.

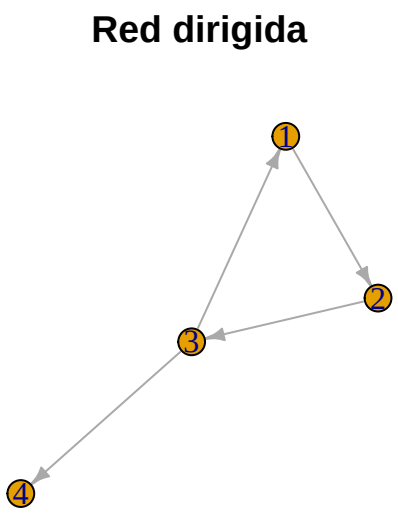
Un hipergrafo $H = (V, E)$ es donde E es una colección de subconjuntos no vacíos de V llamados hiperaristas, que pueden contener más de dos vértices.



Por ejemplo, en bioinformática, los hipergrafos se utilizan para modelar redes de interacción de proteínas. En estas redes, una hiperarista puede representar un conjunto de proteínas que interactúan todas juntas en un complejo molecular. O bien, en una base de datos de una tienda en línea, un hipergrafo puede representar las relaciones entre clientes, productos, y transacciones.

Grafo Dirigido (Directed graph)

Un grafo dirigido o digrafo $D = (V, A)$ tiene aristas dirigidas. Las aristas son pares ordenados de vértices (u, v), donde $u, v \in V$. Los grafos dirigidos, donde las aristas tienen una dirección definida, son esenciales para modelar relaciones asimétricas.



Ejemplo 1: Redes de Comunicación en medios digitales:
En redes como Instagram, los usuarios publican contenido y se conectan con otros usuarios. Aquí, el outdegree de un nodo (usuario) representa el número de publicaciones o mensajes que envía a otros usuarios. Un outdegree alto puede indicar un usuario muy activo o influyente, que inicia muchas conversaciones o interacciones. Analizar el outdegree en este contexto es relevante para identificar a los usuarios más activos, influenciadores clave, o para entender patrones de difusión de información en la red.

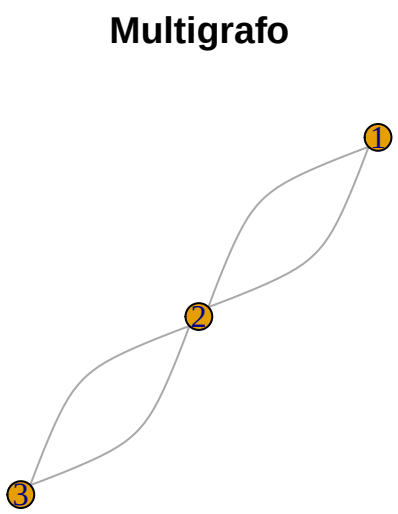
Ejemplo 2: Citas en Publicaciones Científicas:
En el ámbito académico, las citas a artículos científicos son indegree para sus autores (y el artículo en sí). Un indegree alto sugiere una alta relevancia o impacto del artículo en su campo. Estudiar el indegree en este contexto ayuda a identificar los trabajos o autores más influyentes, tendencias en la investigación, y la evolución de diferentes campos científicos.

Nótese que: $\sum_{v \in V} \text{indegree}(v) = \sum_{v \in V} \text{outdegree}(v)$

Multigrafo (Multigraph)

En un multigrafo, a diferencia de un grafo simple, se permite que existan múltiples aristas entre un mismo par de vértices. Esto significa que puede haber más de una conexión directa entre dos vértices dados. En este contexto, cada arista en el multigrafo se define de manera única no solo por los vértices que conecta, sino también por un identificador adicional.

Un multigrafo se define como $M = (V, E)$. Matemáticamente, una arista es un trío (v, w, k) donde $v, w \in V$ y k es un identificador único para la arista (dado que dos nodos pueden estar conectados por más de una arista)



Ejemplo 1, supongamos que queremos analizar empíricamente la siguiente pregunta "¿Cómo afectan las múltiples formas de interacción (amistad, colaboración profesional, intereses compartidos) a la estructura y dinámica de las redes sociales en línea?"

Un multigrafo permite modelar estas múltiples relaciones, donde cada tipo de interacción es representado por una arista diferente entre el mismo par de nodos (personas). La investigación puede centrarse en cómo estas capas de relación se superponen y afectan aspectos como la formación de comunidades, la difusión de información, o la influencia social

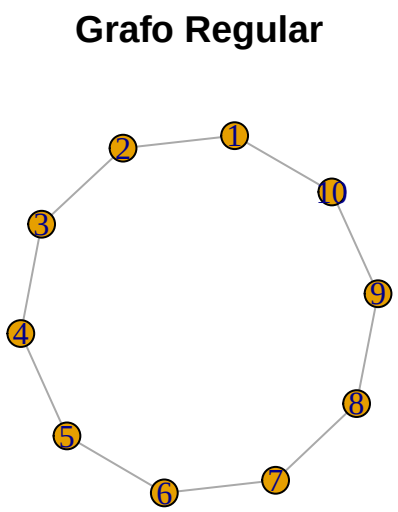
Ejemplo 2, supongamos ahora que la pregunta es "¿De qué manera los distintos canales de comunicación (como discursos públicos, publicaciones en medios de comunicación, y comunicaciones internas) entre políticos y grupos de interés influyen en la formación de políticas públicas?"

En este caso, nuevamente un multigrafo puede ayudar. Cada arista representa un canal de comunicación distinto. El análisis puede revelar patrones en la forma en que la información y la influencia circulan a través de estos canales y cómo esto afecta la toma de decisiones y la formación de políticas.

Patrones en grafos

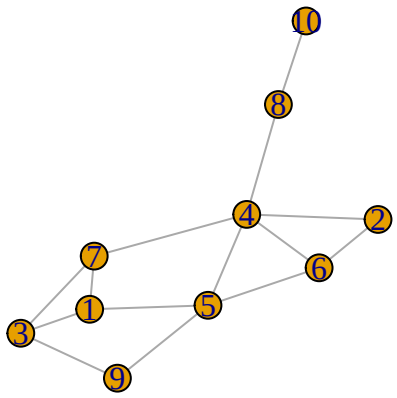
Regularidad

Un grafo regular es aquel donde todos los vértices tienen el mismo grado. Por ejemplo, un grafo completo K_n, donde cada vértice está conectado con todos los demás, y cada vértice tiene un grado de $n-1$. O un grafo circular, que a su vez es un ciclo o recorrido cíclico de Euler, como el siguiente:



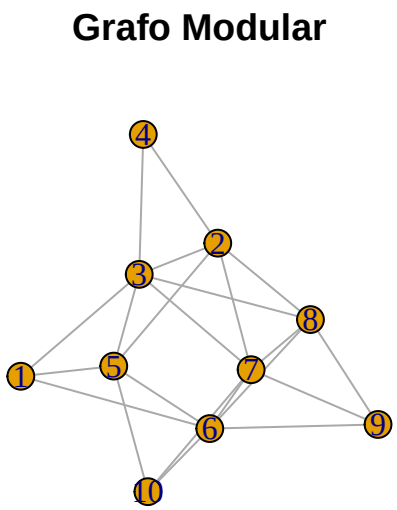
Aleatoriedad

Los grafos aleatorios no presentan una estructura clara de conexiones. El modelo Erdős-Rényi genera grafos donde la presencia de cada arista es independiente y con la misma probabilidad. Lo veremos con detalle más adelante, pero por ahora, es útil constatar que la arista l_{ij} entre los nodos i y j podría existir sólo como una probabilidad. O dicho de otro modo, que podemos entender a la red ya no sólo con un valor dato, sino también con un valor probable. Ello nos abre un espacio para estudiar temas como dinámica en redes, contagio en redes, muestreo en redes, etc. . .



Modularidad

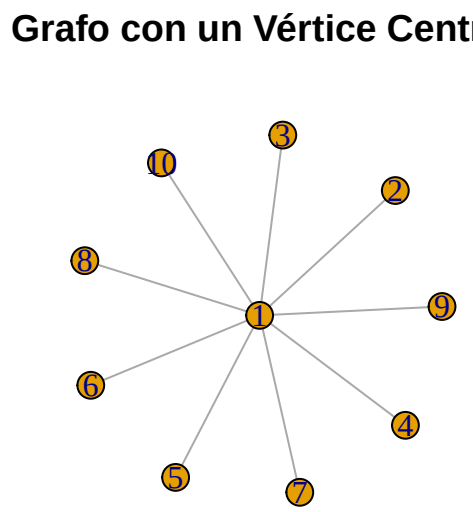
La modularidad indica la presencia de comunidades dentro de la red. Los grafos modulares muestran conexiones densas entre vértices del mismo módulo y conexiones más escasas entre módulos diferentes.



La modularidad se mide generalmente a través de métricas que comparan la densidad de enlaces dentro de los módulos con la densidad de enlaces entre módulos. Un valor alto de modularidad indica una estructura de red claramente dividida en comunidades. Lo veremos con detalle en el módulo 4 del curso.

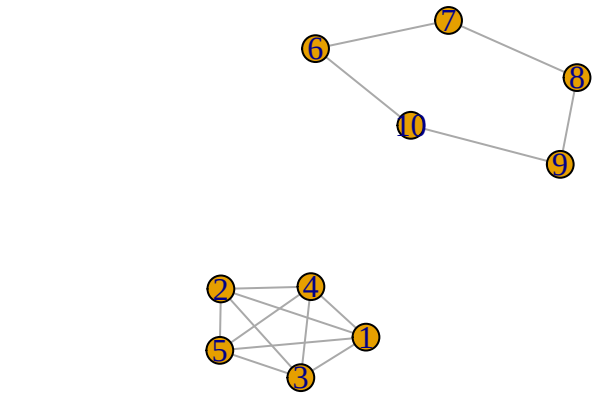
Centralidad

Las medidas de centralidad (que son el objeto del siguiente módulo del curso) son vitales en el análisis de patrones en redes. En términos generales, la centralidad mide la importancia de un vértice dentro de la red. Los individuos con alta centralidad en redes sociales suelen ser líderes de opinión, influenciadores, poderosos, populares, etc. Lo que va cambiando es qué se entiende por "importancia" lo que genera espacio para la creación de un amplio set de medidas de centralidad.



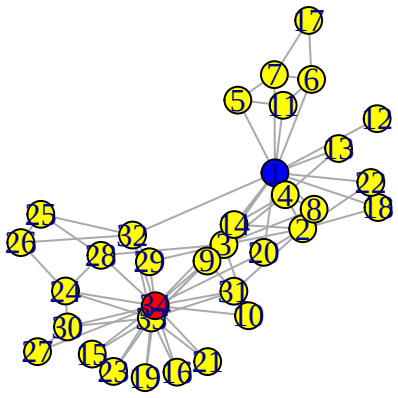
Agrupamiento

El coeficiente de agrupamiento refleja qué tan interconectados están los vecinos de un vértice, formando cliqués o grupos cerrados de amigos.



Homofilia

La homofilia se observa en grafos donde vértices similares tienden a conectarse más entre sí, formando subgrafos densos basados en atributos comunes.



Los patrones en grafos como la regularidad, la aleatoriedad, la modularidad, la centralidad, el agrupamiento y la homofilia son cruciales para entender las dinámicas y estructuras en redes sociales. A través de estos conceptos, podemos identificar líderes, comunidades y patrones de formación de amistades y colaboraciones, etcétera

Matrices de Adyacencia y de Incidencia

La matriz de adyacencia representa la relación entre nodos: La matriz de adyacencia A de un grafo G con n vértices es una matriz $n \times n$ donde a_{ij} es 1 si hay una arista entre los vértices i y j, y 0 en caso contrario.

La matriz de incidencia representa las relaciones entre nodos y aristas. La matriz de incidencia B de un grafo G con n vértices y m aristas es una matriz $n \times m$ donde b_{ij} es 1 si el vértice i incide en la arista j, y 0 en caso contrario.

La matriz de adyacencia es más eficiente para grafos densos, mientras que la matriz de incidencia puede ser más adecuada para grafos dispersos.

Listas de Adyacencia

La lista de adyacencia es otra forma de representar un grafo, donde cada vértice tiene una lista de otros vértices a los que está conectado.

Propiedades y Medidas

Grado

Ya lo vimos anteriormente, el grado de un vértice en un grafo es el número de aristas que inciden en él. Para grafos dirigidos, distinguimos entre grado de entrada y grado de salida.

$$\deg(v) = \text{número de aristas incidentes en } v$$

En un grafo dirigido, distinguimos entre el grado de entrada y el grado de salida:

$$\deg_{\text{entrada}}(v) = \text{número de aristas entrantes en } v$$

$$\deg_{\text{salida}}(v) = \text{número de aristas salientes de } v$$

Densidad

La densidad de un grafo se define como el número de aristas sobre el número máximo posible de aristas entre vértices.

Para un grafo con n vértices:

$$\text{Densidad} = \frac{\text{número de aristas actuales}}{\text{número máximo de aristas posibles}} = \frac{2m}{n(n-1)} \text{ para un grafo no dirigido}$$

$$= \frac{m}{n(n-1)} \text{ para un grafo dirigido}$$

Donde m es el número total de aristas.

Distancia y Diámetro

La distancia entre dos vértices es el número mínimo de aristas que se deben atravesar para ir de un vértice a otro. El diámetro de un grafo es la mayor distancia entre cualquier par de vértices.

La distancia entre dos vértices u y v:

$$d(u, v) = \text{mínimo número de aristas para ir de } u \text{ a } v$$

El diámetro de un grafo es la mayor distancia entre cualquier par de vértices:

$$\text{Diámetro} = \max\{d(u, v) : u, v \in V, u \neq v\}$$

Donde V es el conjunto de vértices del grafo.

Transitividad

La transitividad, o coeficiente de agrupamiento global, es una medida de la proporción de triángulos en el grafo.

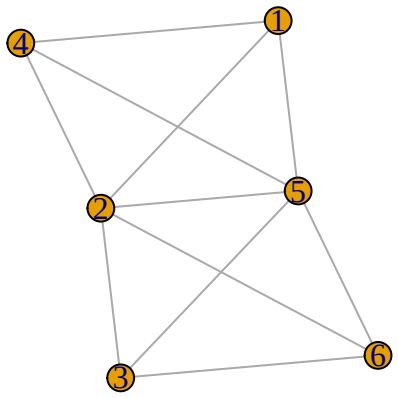
La transitividad, o coeficiente de agrupamiento global, se define como:

$$T = \frac{3 \times \text{número de triángulos en el grafo}}{\text{número de tríadas conectadas}}$$

Donde una tríada conectada es un conjunto de tres vértices donde al menos hay dos aristas entre ellos.

Ejercicios para la casa

Calcular las propiedades del siguiente grafo simple.



Veamos ahora operaciones con redes en entorno R.

```
rm(list=ls())
require(here)
```

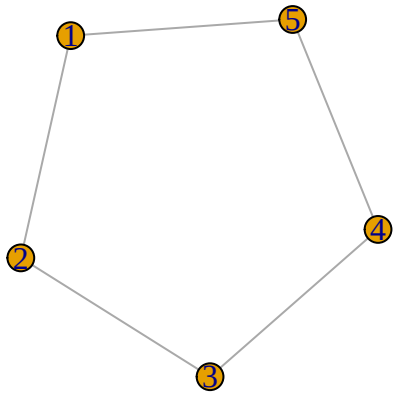
```
## Loading required package: here
```

```
## here() starts at D:/CURSOS/RedesSociales_doctorado/teoriaredes
```

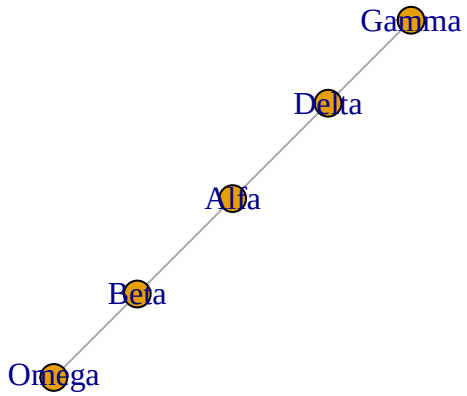
```
library(igraph)
library(intergraph)
```

```
## Warning: package 'intergraph' was built under R version 4.3.2
```

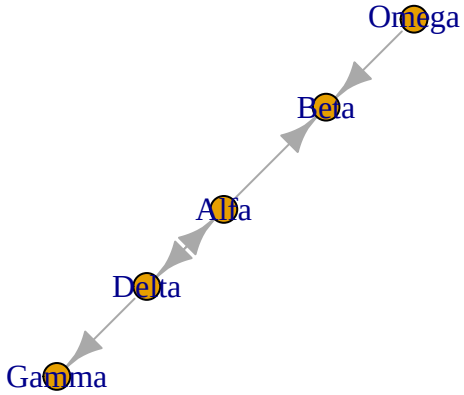
```
aqui <- here()



################################################################################
# Una red se puede generar de varias formas.
################################################################################
# 1a. Juntando un conjunto de links representados como números o conceptos

red1 <- graph_from_literal(1-2,
                           2-3,
                           3-4,
                           4-5,
                           1-5,
                           2-3)
plot(red1)
```



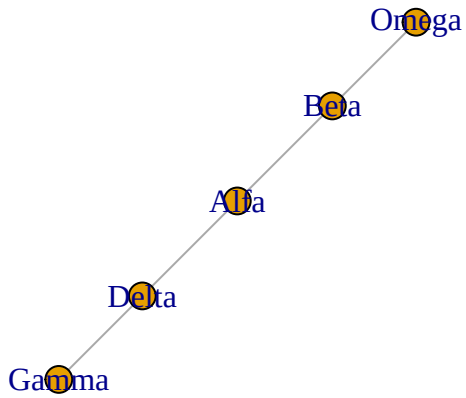
```
red2 <- graph_from_literal(Alfa-Beta, # en este caso toma los nombres como factores
                           Gamma-Delta,
                           Delta-Alfa,
                           Omega-Beta)
plot(red2)
```



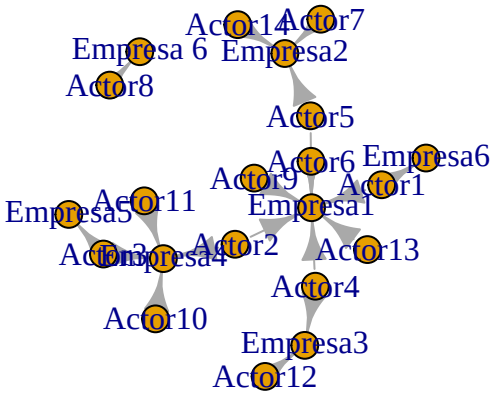

```
red3 <- graph_from_literal(Alfa-+Beta, # aquí es digida
                           Gamma+-Delta,
                           Delta++Alfa,
                           Omega-+Beta)
plot(red3)
```



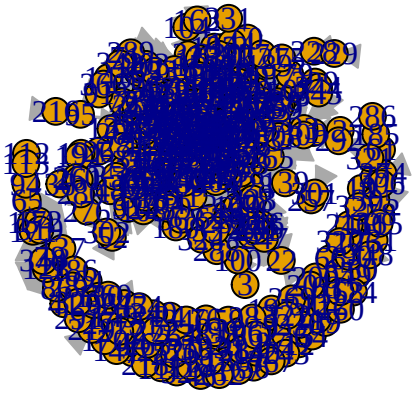

```
# 1b se puede combinar un edgelist con atributos

relaciones <- matrix(c("Alfa","Beta",  #edlist
                 "Gamma","Delta",
                 "Delta","Alfa",
                 "Omega","Beta"), ncol=2, byrow = T)
atributos <- matrix(c("Alfa","Beta","Gamma","Delta","Omega", #atributos
                    "4","4","5","5","5", # letras
                    "2","1","2","1","1"), # veces que tienen la letra "a"
                    ncol=3, nrow = 5, byrow=F)
red4 <- graph_from_data_frame(relaciones,
                              vertices=atributos,
                              directed = F)
plot(red4)
```




```
# 1c se puede importar en diversos formatos
x <- read.csv(paste0(aqui,"/data/red_ilustracion.csv"), sep=";")
relaciones5 <- cbind(x$nombre,x$empresa)
red5 <- graph_from_data_frame(relaciones5,
                              directed = T)
plot(red5)
```




```
# 1d importar la red desde algun formato de red
red6 <- read_graph(paste0(aqui,"/data/red_empresas.igraph"))
plot(red6)
```



```
##############################################################################
# Caracteristicas del objeto igraph
##############################################################################

red2$name <- "Grafo en clase"

# Vertices
V(red2)
```

```
## + 5/5 vertices, named, from ef6970c:
## [1] Alfa  Beta  Gamma Delta Omega
```

```
# Edges
E(red2)
```

```
## + 4/4 edges from ef6970c (vertex names):
## [1] Alfa --Beta  Alfa --Delta Beta --Omega Gamma--Delta
```

```
# ambos datos
print_all(red2) # o simplemente
```

```
## IGRAPH ef6970c UN-- 5 4 -- Grafo en clase
## + attr: name (g/c), name (v/c)
## + edges from ef6970c (vertex names):
## [1] Alfa --Beta  Alfa --Delta Beta --Omega Gamma--Delta
```

```
red2
```

```
## IGRAPH ef6970c UN-- 5 4 -- Grafo en clase
## + attr: name (g/c), name (v/c)
## + edges from ef6970c (vertex names):
## [1] Alfa --Beta  Alfa --Delta Beta --Omega Gamma--Delta
```

```
# atributos de los vertices y edges
V(red2)$name # ya estaban
```

```
## [1] "Alfa"  "Beta"  "Gamma" "Delta" "Omega"
```

```
E(red2)$name # no hay
```

```
## NULL
```

```
E(red2)$name <- c("a","b","c","d")

V(red2)$color <- c("blue","red","red","red","pink")

##############################################################################
# Operaciones con el objeto igraph
##############################################################################

class(red2)
```

```
## [1] "igraph"
```

```
is_weighted(red2)
```

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

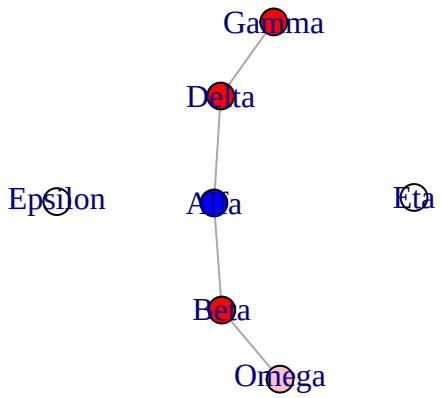
```
is_simple(red2)
```

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

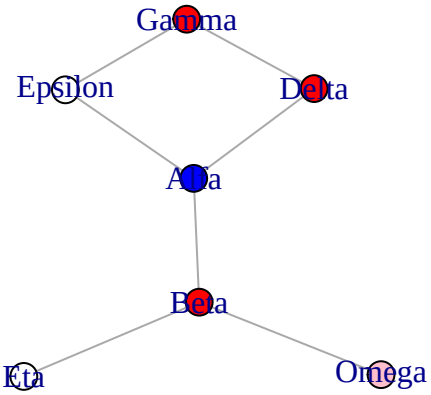
```
# Se puede trabajar con subgrafos de la red

red2_a <- induced_subgraph(red2, 1:3) # los nodos según el orden dado
print_all(red2_a)
```

```
## IGRAPH efb0ad2 UN-- 3 1 -- Grafo en clase
## + attr: name (g/c), name (v/c), color (v/c), name (e/c)
## + edge from efb0ad2 (vertex names):
## [1] Alfa--Beta
```

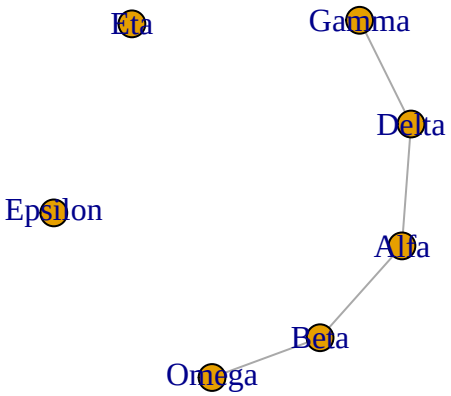
```
red2_b <- red2 - vertices(c(1)) # se lee, red2_b es la red2 menos los vértices indicados

# Se pueden agregar nodos y edges
red2_c <- red2 + vertices(c("Epsilon","Eta"))
plot(red2_c)
```




```
red2_d <- red2_c + edges(c("Epsilon","Gamma"),
                         c("Eta","Beta"),
                         c("Alfa","Epsilon"))
plot(red2_d)
```



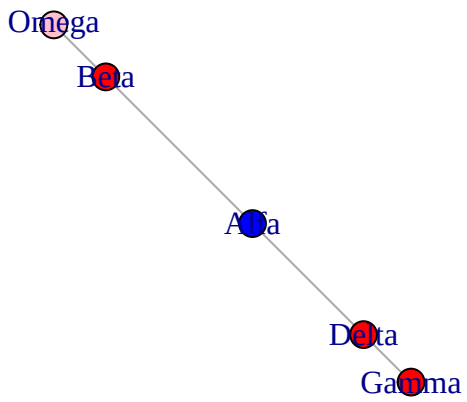

```
# o unir redes
red2_e <- red2 + red2_c
plot(red2_e)
```



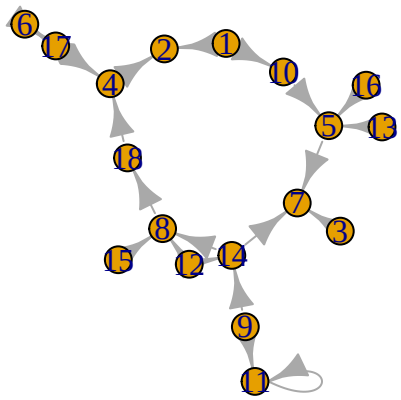
```r
# podemos crear una red con pesos a partir de la anterior
red2_f <- red2
E(red2_f)$weight <- c(0.1,0.2,1,1.3)
is_weighted(red2_f)
```

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

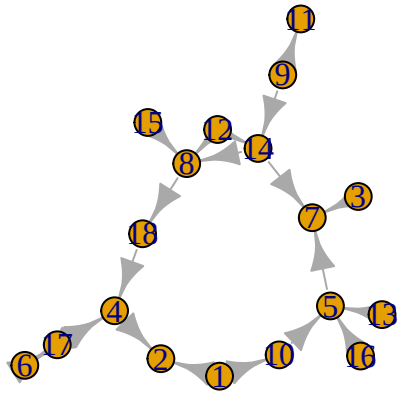
```r
plot(red2_f)
```



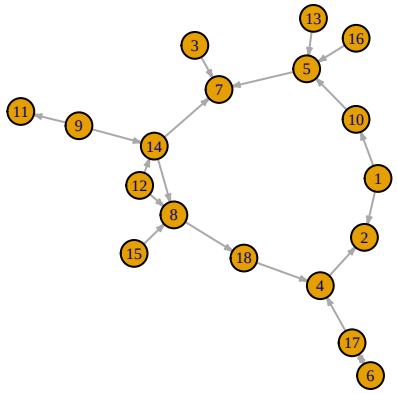

```r
##############################################################################
# visualizando la red
##############################################################################

# ahora una red un poco mas grande
nodos <- seq(1:18)
links <- matrix(c(1,12,3,14,5,6,17,8,9,1,10,4,
                  12,13,14,15,16,17,18,9,11,10,14,7,
                  7,7,17,6,18,14,2,5,2,8,5,8,8,5,
                  4,4,11,11),nrow=21,ncol=2)

red2 <- graph_from_data_frame(d=links, vertices=nodos, directed=T)
plot(red2)
```



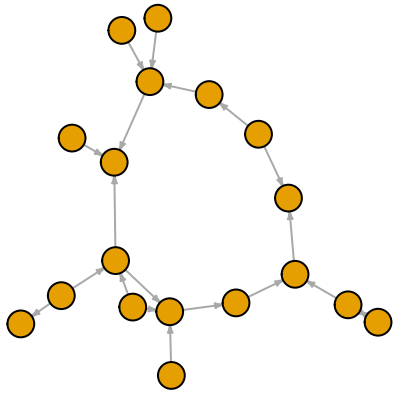

```
red2 <- simplify(red2, remove.multiple = F, remove.loops = T)
plot(red2)
```



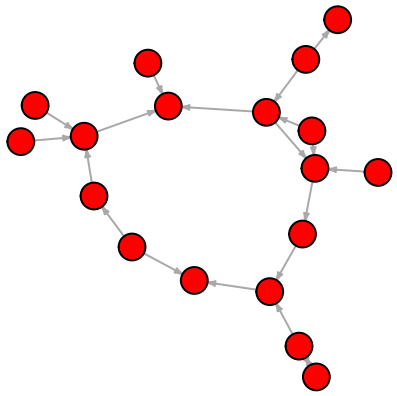

```
# se puede cambiar el tamaño de la flecha, de las etiquetas, Cambiar colores, etc
plot(red2, edge.arrow.size=.2,vertex.label.cex=0.5)
```



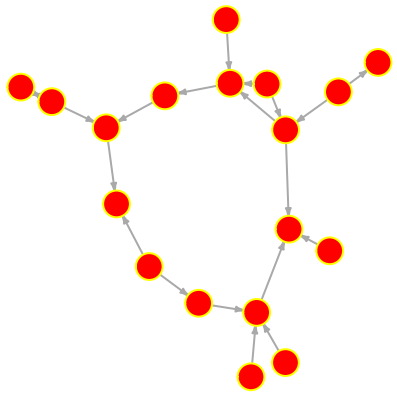

```
plot(red2, edge.arrow.size=.2,vertex.label=NA)
```



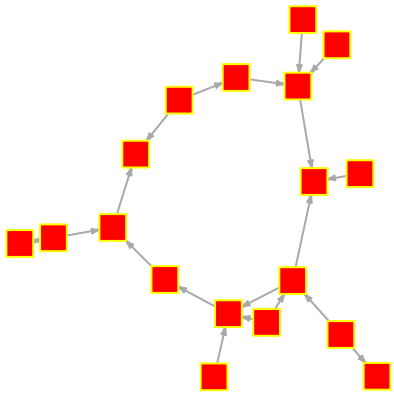
```
plot(red2, edge.arrow.size=.2,vertex.label=NA, vertex.color="red")
```



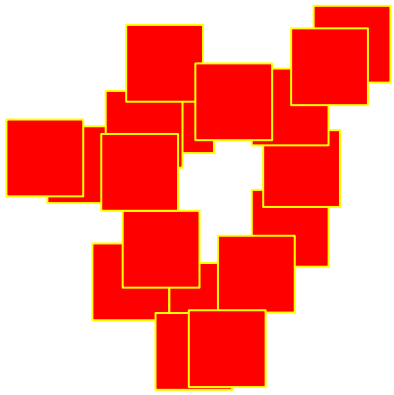
```
plot(red2, edge.arrow.size=.2,vertex.label=NA, vertex.color="red",vertex.frame.color="yellow")
```



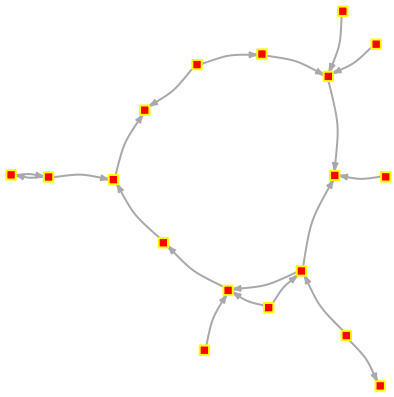
```
plot(red2, edge.arrow.size=.2,vertex.label=NA, vertex.color="red",
     vertex.frame.color="yellow",vertex.shape="square")
```



```
plot(red2, edge.arrow.size=.2,vertex.label=NA, vertex.color="red",
     vertex.frame.color="yellow",vertex.shape="square", vertex.size=50)
```

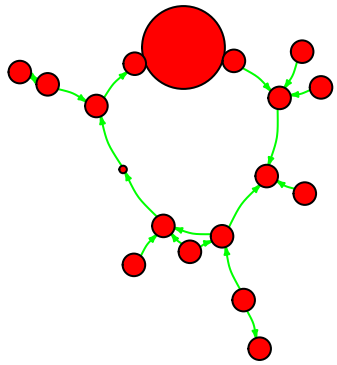


```
plot(red2, edge.arrow.size=.2,vertex.label=NA, vertex.color="red",
     vertex.frame.color="yellow",vertex.shape="square", vertex.size=5,edge.curved=.2)
```

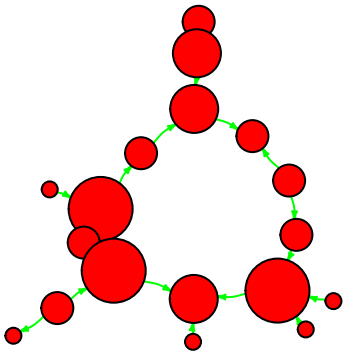


```
# cambiando los tamaños de los nodos de a uno
size_nodo <- c(55,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,5)
```

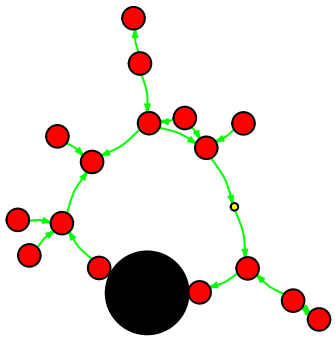
```
plot(red2, edge.arrow.size=.2,vertex.label=NA, vertex.color="red",
     vertex.frame.color="black",vertex.shape="circle", edge.curved=.2,
     edge.color="green", vertex.size=size_nodo)
```



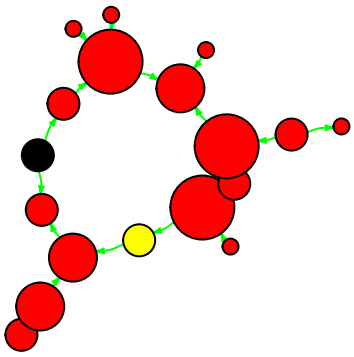
```
b <- igraph::degree(red2)
plot(red2, edge.arrow.size=.2,vertex.label=NA, vertex.color="red",
     vertex.frame.color="black",vertex.shape="circle", edge.curved=.2,
     edge.color="green", vertex.size=b*10)
```



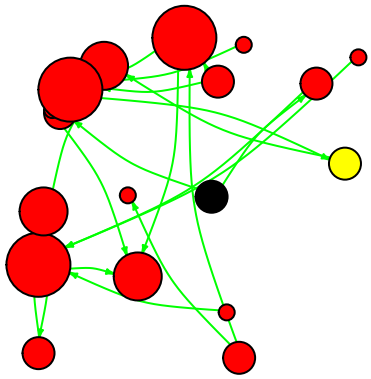
```
# misma lógica para cambiar colores, por ejemplo
color_nodo <- c("black","red","red","red","red","red","red","red","red","red",
                "red","red","red","red","red","red","red","yellow")
plot(red2, edge.arrow.size=.2,vertex.label=NA, vertex.color=color_nodo,
     vertex.frame.color="black",vertex.shape="circle", edge.curved=.2,
     edge.color="green", vertex.size=size_nodo)
```



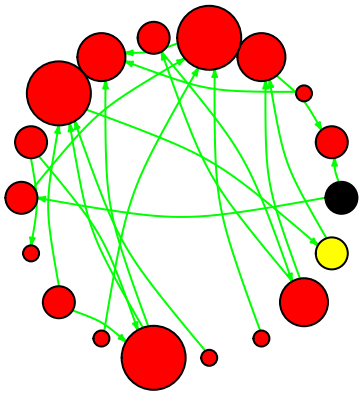
```
# cada vector puede a su vez ser una funcion de algo. Por ejemplo de grado.
size_nodo <- igraph::degree(red2)
plot(red2, edge.arrow.size=.2,vertex.label=NA, vertex.color=color_nodo,
     vertex.frame.color="black",vertex.shape="circle", edge.curved=.2,
     edge.color="green", vertex.size=size_nodo*10)
```



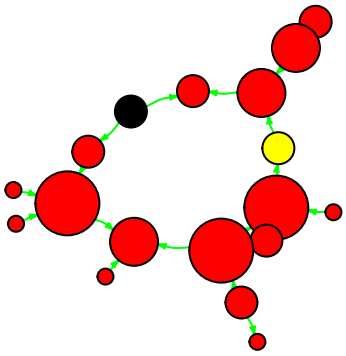
```
###############
# Layouts

plot(red2, edge.arrow.size=.2,vertex.label=NA, vertex.color=color_nodo,
     vertex.frame.color="black",vertex.shape="circle", edge.curved=.2,
     edge.color="green", vertex.size=size_nodo*10,
     layout=layout_randomly)
```



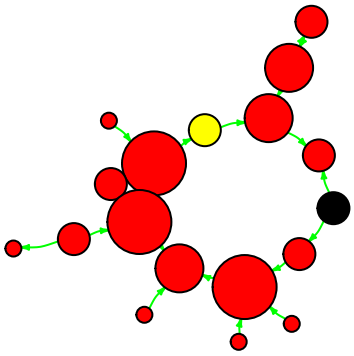
```
l.circulo <- layout_in_circle(red2) # podemos hacer c?lculos previos y luego ponerlos en el plot
plot(red2, edge.arrow.size=.2,vertex.label=NA, vertex.color=color_nodo,
     vertex.frame.color="black",vertex.shape="circle", edge.curved=.2,
     edge.color="green", vertex.size=size_nodo*10,
     layout=l.circulo)
```



```
# Fruchterman-Reingold
l.fr <- layout_with_fr(red2)
plot(red2, edge.arrow.size=.2,vertex.label=NA, vertex.color=color_nodo,
     vertex.frame.color="black",vertex.shape="circle", edge.curved=.2,
     edge.color="green", vertex.size=size_nodo*10,
     layout=l.fr)
```



```
# Kamada Kawai
l.kk <- layout_with_kk(red2)
plot(red2, edge.arrow.size=.2,vertex.label=NA, vertex.color=color_nodo,
     vertex.frame.color="black",vertex.shape="circle", edge.curved=.2,
     edge.color="green", vertex.size=size_nodo*10,
     layout=l.kk)
```



```
layouts <- grep("^layout_", ls("package:igraph"), value=TRUE)[-1]
layouts
```

```
##  [1] "layout_as_bipartite"  "layout_as_star"       "layout_as_tree"
##  [4] "layout_components"    "layout_in_circle"     "layout_nicely"
##  [7] "layout_on_grid"       "layout_on_sphere"     "layout_randomly"
## [10] "layout_with_dh"       "layout_with_drl"      "layout_with_fr"
## [13] "layout_with_gem"      "layout_with_graphopt" "layout_with_kk"
## [16] "layout_with_lgl"      "layout_with_mds"      "layout_with_sugiyama"
```

```
# Ver http://kateto.net/network-visualization para diferentes variaciones para la visualizacion

################################################################################
# redes bimodales
################################################################################

# redes bipartitas
red_b <- graph_from_literal(Alfa-LET,
                            Perro-ANI,
                            Beta-LET,
                            Gato-ANI,
                            Gamma-LET)
# asignemos categorias a cada nodo
V(red_b)$type <- c(FALSE,TRUE,FALSE,TRUE,FALSE,FALSE,FALSE)
print_all(red_b, v=T)
```

```
## IGRAPH efd5a51 UN-B 7 5 --
## + attr: name (v/c), type (v/l)
## + vertex attributes:
## |      name  type
```

```
## | [1]  Alfa FALSE
## | [2]   LET  TRUE
## | [3] Perro FALSE
## | [4]   ANI  TRUE
## | [5]  Beta FALSE
## | [6]  Gato FALSE
## | [7] Gamma FALSE
## + edges from efd5a51 (vertex names):
## [1] Alfa --LET   LET  --Beta  LET  --Gamma Perro--ANI   ANI  --Gato
```

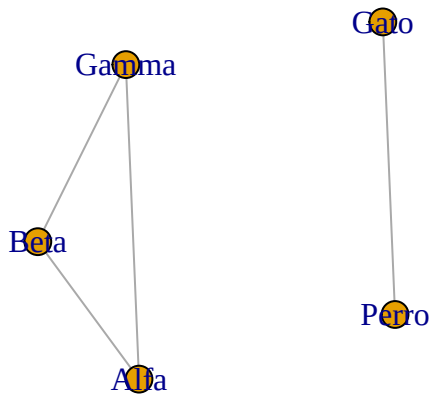
```
red_b_proyecciones <- bipartite_projection(red_b)

red_b_proyecciones$proj1  # proyeccion en nombres
```

```
## IGRAPH efd817a UNW- 5 4 --
## + attr: name (v/c), weight (e/n)
## + edges from efd817a (vertex names):
## [1] Alfa --Beta  Alfa --Gamma Perro--Gato  Beta --Gamma
```

```
red_b_proyecciones$proj2  # proyección en paises
```

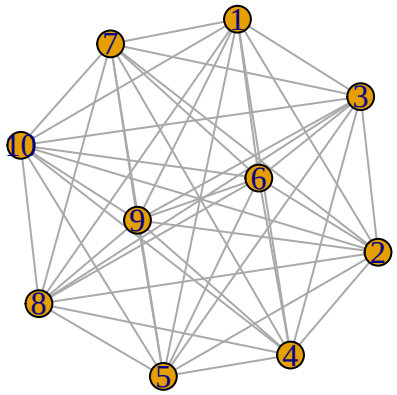
```
## IGRAPH efd81b9 UNW- 2 0 --
## + attr: name (v/c), weight (e/n)
## + edges from efd81b9 (vertex names):
```

```
plot(red_b_proyecciones$proj1)
```

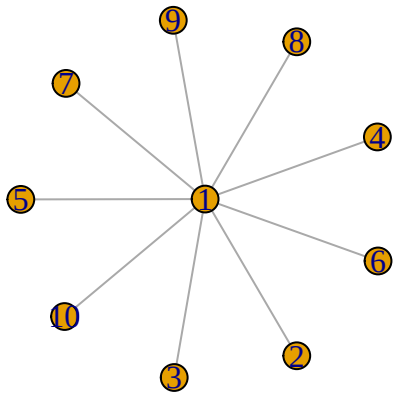



```
plot(red_b_proyecciones$proj2)
```

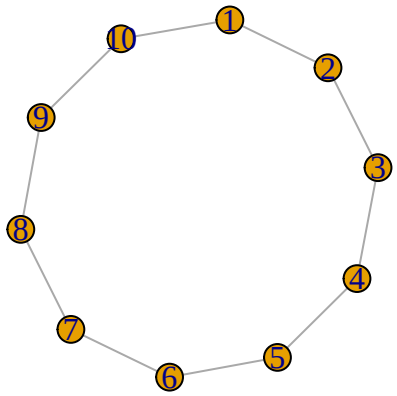
```
##############################################################################
# Grupos especiales de redes
##############################################################################

# red completa
red_completa <- make_full_graph(10, directed = F)
plot(red_completa)
```



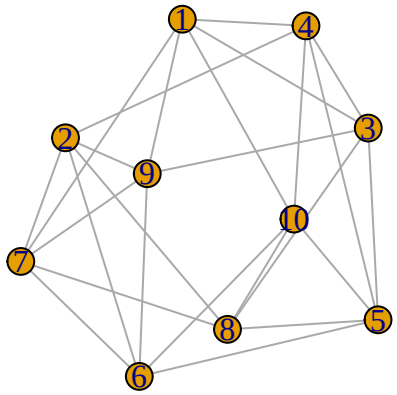

```
# red estrella
red_estrella <- make_star(10, mode="undirected")
plot(red_estrella)
```



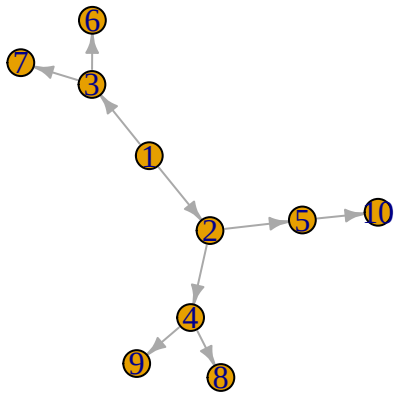
```r
# redes regulares
red_anillo <- make_ring(10, directed = F)
plot(red_anillo)
```




```r
red_regular <- sample_k_regular(10, k=5, directed = FALSE, multiple = FALSE)
plot(red_regular)
```




```
# redes árboles
red_arbol <- make_tree(10, mode = "out")
plot(red_arbol, edge.arrow.size=0.5)
```



```
##############################################################################
# Propiedades de la red
##############################################################################

# genera la matriz de adyacencia
as_adjacency_matrix(red2)
```

```
## 18 x 18 sparse Matrix of class "dgCMatrix"
##   [[ suppressing 18 column names '1', '2', '3' ... ]]
##
## 1  . 1 . . . . . . 1 . . . . . . . .
## 2  . . . . . . . . . . . . . . . . .
## 3  . . . . . . 1 . . . . . . . . . .
## 4  . 1 . . . . . . . . . . . . . . .
## 5  . . . . . . 1 . . . . . . . . . .
## 6  . . . . . . . . . . . . . . . . 1 .
## 7  . . . . . . . . . . . . . . . . .
## 8  . . . . . . . . . . . . . . . . 1
## 9  . . . . . . . . . 1 . . 1 . . . .
## 10 . . . . 1 . . . . . . . . . . . .
## 11 . . . . . . . . . . . . . . . . .
## 12 . . . . . . . 1 . . . . . 1 . . . .
## 13 . . . . 1 . . . . . . . . . . . .
## 14 . . . . . . . 1 1 . . . . . . . .
## 15 . . . . . . . . 1 . . . . . . . .
## 16 . . . . 1 . . . . . . . . . . . .
## 17 . . . 1 . 1 . . . . . . . . . . .
## 18 . . . 1 . . . . . . . . . . . . .
```

```
# analiza la ego network del nodo especificado
neighbors(red2,2)
```

```
## + 0/18 vertices, named, from efb9a60:
```

```
neighbors(red2,1)
```

```
## + 2/18 vertices, named, from efb9a60:
## [1] 2  10
```

```
# grado de la red  (NOTA: require(sna) y ver qué pasa)
igraph::degree(red2)
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18
##  2  2  1  3  4  2  3  4  2  2  1  2  1  4  1  1  3  2
```

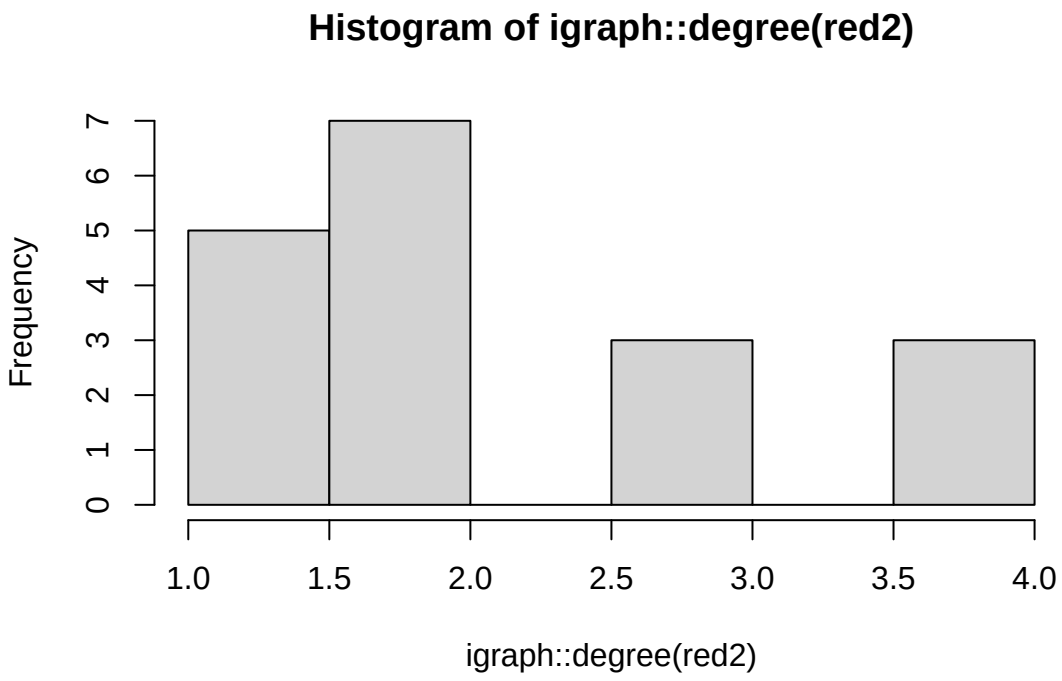
```
igraph::degree(red2, mode="in")
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18
##  0  2  0  2  3  1  3  3  0  1  1  0  0  2  0  0  1  1
```

```
igraph::degree(red2, mode="out")
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18
##  2  0  1  1  1  1  0  1  2  1  0  2  1  2  1  1  2  1
```

```
hist(igraph::degree(red2))
```




```
# cuando es una red con pesos se usa strength(_nombrered_)

# caminos y conectividad en la red
# completa o incompleta
is_connected(red2)
```

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

```
decompose(red2) # muestra los componentes
```

```
## [[1]]
## IGRAPH f074d40 DN-- 18 20 --
## + attr: name (v/c)
## + edges from f074d40 (vertex names):
##  [1] 1 ->10 12->14 3 ->7  14->7  5 ->7  6 ->17 17->6  8 ->18 9 ->14 1 ->2
## [11] 10->5  4 ->2  12->8  13->5  14->8  15->8  16->5  17->4  18->4  9 ->11
```

```
table(sapply(decompose(red2), vcount)) # tabla resumen de los componentes
```

```
##
## 18
##  1
```

```
# una medida de conectividad interesante es saber si hay puntos vulnerables que
# de no existir se rompe la red:

articulation_points(red2)
```

```
## + 7/18 vertices, named, from efb9a60:
## [1] 17 4  7  5  9  14 8
```

```r
# para ver lo mismo en una red con más nodos, usemos el paquete
# igraphdata que contiene una colección de grafos

require(igraphdata)
```

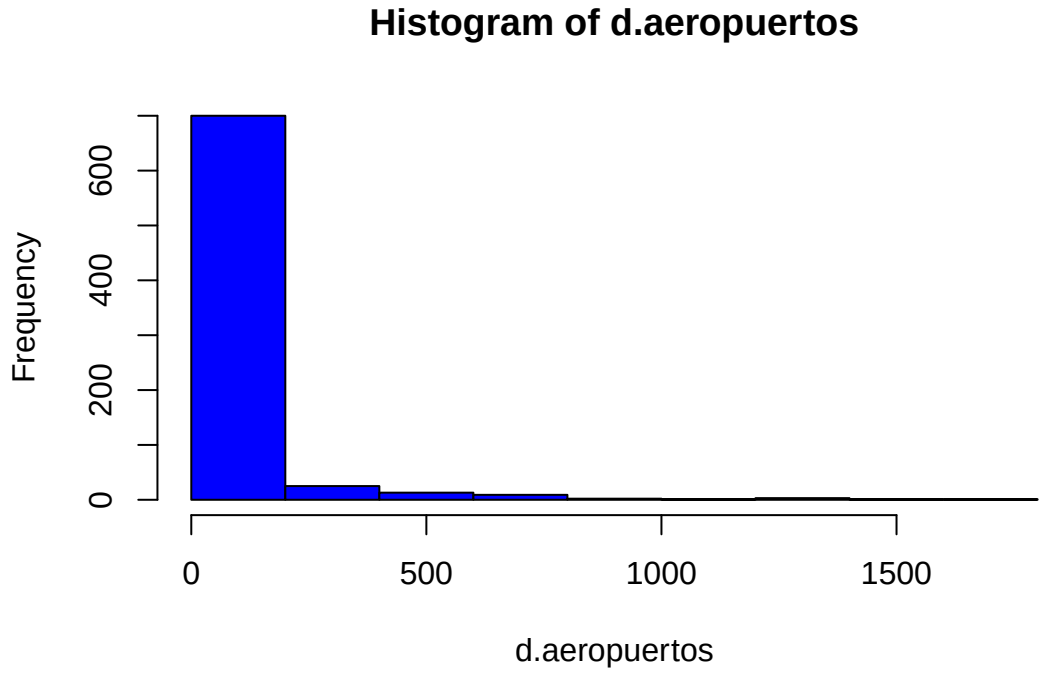
```
## Loading required package: igraphdata
```

```
## Warning: package 'igraphdata' was built under R version 4.3.2
```

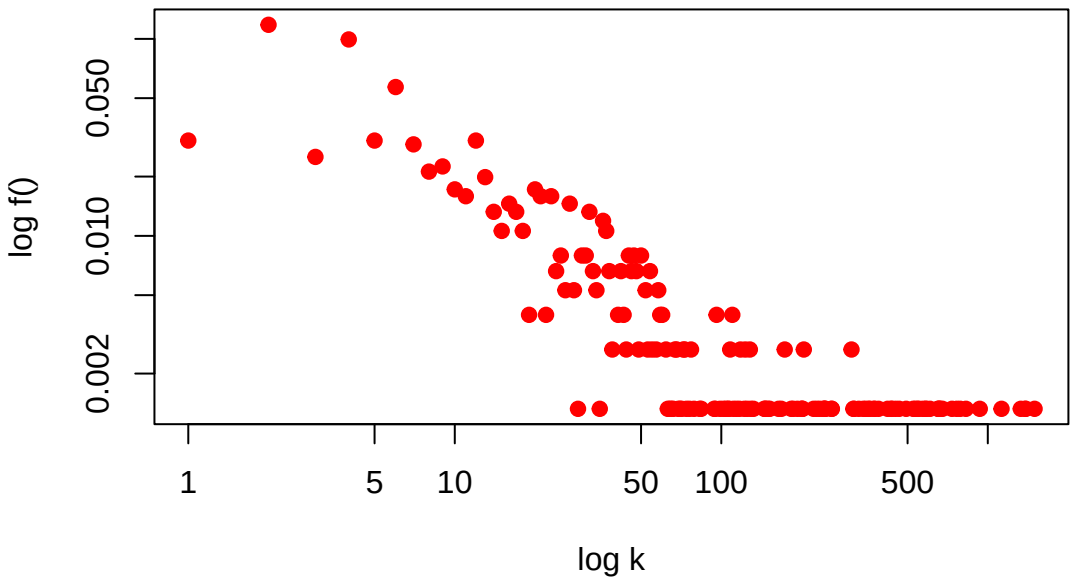
```r
data(package="igraphdata")

data(USairports)
d.aeropuertos <- igraph::degree(USairports)
```

```
## This graph was created by an old(er) igraph version.
##   Call upgrade_graph() on it to use with the current igraph version
##   For now we convert it on the fly...
```

```r
hist(d.aeropuertos, col="blue")
```

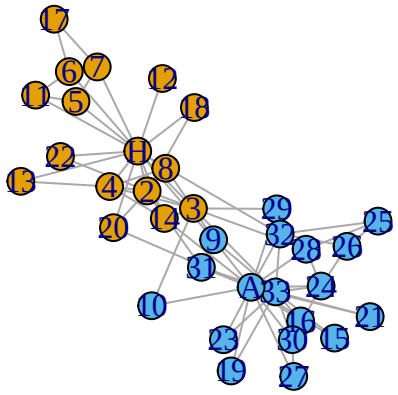



```r
# por la rapidez con que decae la distribución es mejor usar log-log
dist.aeropuertos <- degree_distribution(USairports)
d <- 1:max(d.aeropuertos) - 1 # generamos una secuencia que la usaremos en el gráfico
val <- (dist.aeropuertos != 0) # creamos un vector lógico de valores con grado !=0
plot(d[val],dist.aeropuertos[val],
     log="xy",
     col="red",
     ylab="log f()",
     xlab="log k",
     pch = 19)
```




```
# cliques
# Estos algoritmos son lentos porque son censos de toda la red karate
data(karate)
plot(karate)
```

```
## This graph was created by an old(er) igraph version.
##   Call upgrade_graph() on it to use with the current igraph version
##   For now we convert it on the fly...
```



```
# así que lo haremos con la red2
# te entrega una tabla resumen de la cantidad de cliques de tamaño 1, 2, 3,...
table(sapply(cliques(karate), length))
```

```
##
##  1  2  3  4  5
## 34 78 45 11  2
```

```
# tambien se puede buscar el clique más grande, denominado "the clique number"
clique_num(karate)
```

```
## [1] 5
```

```
# k-cores
cores <- coreness(karate) # entrega el grado máximo del core en que puede estar cada nodo

require(sna)
```

```
## Loading required package: sna
```

```
## Warning: package 'sna' was built under R version 4.3.1
```

```
## Loading required package: statnet.common
```

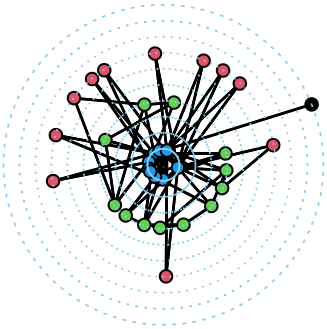
```
## Warning: package 'statnet.common' was built under R version 4.3.1
```

```
##
## Attaching package: 'statnet.common'
```

```
## The following objects are masked from 'package:base':
##
##     attr, order
```

```
## Loading required package: network
```

```
## Warning: package 'network' was built under R version 4.3.1
##
## 'network' 1.18.1 (2023-01-24), part of the Statnet Project
## * 'news(package="network")' for changes since last version
## * 'citation("network")' for citation information
## * 'https://statnet.org' for help, support, and other information
##
## Attaching package: 'network'
## The following objects are masked from 'package:igraph':
##
##     %c%, %s%, add.edges, add.vertices, delete.edges, delete.vertices,
##     get.edge.attribute, get.edges, get.vertex.attribute, is.bipartite,
##     is.directed, list.edge.attributes, list.vertex.attributes,
##     set.edge.attribute, set.vertex.attribute
## sna: Tools for Social Network Analysis
## Version 2.7-1 created on 2023-01-24.
## copyright (c) 2005, Carter T. Butts, University of California-Irvine
##  For citation information, type citation("sna").
##  Type help(package="sna") to get started.
##
## Attaching package: 'sna'
## The following objects are masked from 'package:igraph':
##
##     betweenness, bonpow, closeness, components, degree, dyad.census,
##     evcent, hierarchy, is.connected, neighborhood, triad.census
require(intergraph)
karate.sna <- intergraph::asNetwork(karate)
sna::gplot.target(karate.sna,cores,circ.lab=F,
                  circ.col="skyblue",usearrows=F,
                  vertex.col=cores) # mientras más al medio + alto el k-core
```



```
detach("package:sna") # porque tiene funciones con nombres iguales que igraph

# dyads
dyad_census(karate)
```

```
## Warning: `dyad_census()` requires a directed graph.
```

```
## $mut
## [1] 78
##
## $asym
## [1] 0
##
## $null
## [1] 483
```

```
# triads --> motifs
triad_census(karate)  # usamos karate porque es una red dirigida (i.e. más variedad)
```

```
## Warning in triad_census(karate): At core/misc/motifs.c:1165 : Triad census
## called on an undirected graph.
```

```
##  [1] 3971    0 1575    0    0    0    0    0    0    0  393    0    0    0    0
## [16]   45
```

```
# para ver los códigos de cada motif https://igraph.org/r/doc/triad_census.html

transitivity(karate) # clustering de la red (# tríadas/ # potencial tríadas)
```

```
## [1] 0.2556818
```

```r
reciprocity(karate) # como es una red no dirigida va a ser 1 siempre
```

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

```r
reciprocity(red2)
```

```
## [1] 0.1
```

```r
# clusters
clusters(karate)
```

```
## $membership
##    Mr Hi  Actor 2  Actor 3  Actor 4  Actor 5  Actor 6  Actor 7  Actor 8
##        1        1        1        1        1        1        1        1
##  Actor 9 Actor 10 Actor 11 Actor 12 Actor 13 Actor 14 Actor 15 Actor 16
##        1        1        1        1        1        1        1        1
## Actor 17 Actor 18 Actor 19 Actor 20 Actor 21 Actor 22 Actor 23 Actor 24
##        1        1        1        1        1        1        1        1
## Actor 25 Actor 26 Actor 27 Actor 28 Actor 29 Actor 30 Actor 31 Actor 32
##        1        1        1        1        1        1        1        1
## Actor 33   John A
##        1        1
##
## $csize
## [1] 34
##
## $no
## [1] 1
```

```r
diameter(karate, weights=NA)
```

```
## [1] 5
```

```r
diameter(USairports, weights=NA)
```

```
## [1] 9
```

```r
################################################################################
################################################################################
################################################################################
# red de los medici
################################################################################
################################################################################
################################################################################

library(network) # network, sna, igraph
data(flo) # es una matriz de adyacencia
medici <- igraph::graph_from_adjacency_matrix(flo,mode = "undirected")

###############
# el objeto igraph
# la clase igraph entrega info basica
medici
```

```
## IGRAPH f55acc1 UN-- 16 20 --
## + attr: name (v/c)
## + edges from f55acc1 (vertex names):
##  [1] Acciaiuoli--Medici       Albizzi   --Ginori       Albizzi   --Guadagni
```

```
##  [4] Albizzi   --Medici       Barbadori --Castellani   Barbadori --Medici
##  [7] Bischeri  --Guadagni     Bischeri  --Peruzzi      Bischeri  --Strozzi
## [10] Castellani--Peruzzi      Castellani--Strozzi      Guadagni  --Lamberteschi
## [13] Guadagni  --Tornabuoni   Medici    --Ridolfi      Medici    --Salviati
## [16] Medici    --Tornabuoni   Pazzi     --Salviati     Peruzzi   --Strozzi
## [19] Ridolfi   --Strozzi      Ridolfi   --Tornabuoni
```

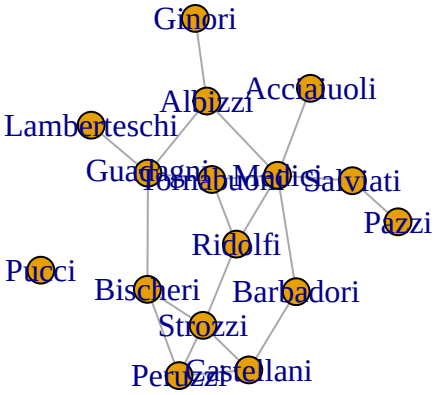
```
E(medici)
```

```
## + 20/20 edges from f55acc1 (vertex names):
##  [1] Acciaiuoli--Medici       Albizzi   --Ginori       Albizzi   --Guadagni
##  [4] Albizzi   --Medici       Barbadori --Castellani   Barbadori --Medici
##  [7] Bischeri  --Guadagni     Bischeri  --Peruzzi      Bischeri  --Strozzi
## [10] Castellani--Peruzzi      Castellani--Strozzi      Guadagni  --Lamberteschi
## [13] Guadagni  --Tornabuoni   Medici    --Ridolfi      Medici    --Salviati
## [16] Medici    --Tornabuoni   Pazzi     --Salviati     Peruzzi   --Strozzi
## [19] Ridolfi   --Strozzi      Ridolfi   --Tornabuoni
```

```
V(medici)
```

```
## + 16/16 vertices, named, from f55acc1:
##  [1] Acciaiuoli   Albizzi      Barbadori    Bischeri     Castellani
##  [6] Ginori       Guadagni     Lamberteschi Medici       Pazzi
## [11] Peruzzi      Pucci        Ridolfi      Salviati     Strozzi
## [16] Tornabuoni
```

```
plot(medici)
```




```
as_edgelist(medici, names=T)
```

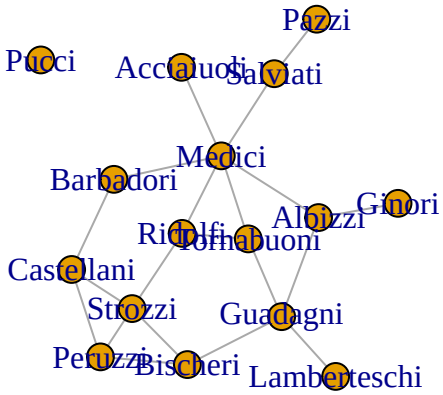
```
##       [,1]         [,2]
##  [1,] "Acciaiuoli" "Medici"
```

```
##  [2,] "Albizzi"    "Ginori"
##  [3,] "Albizzi"    "Guadagni"
##  [4,] "Albizzi"    "Medici"
##  [5,] "Barbadori"  "Castellani"
##  [6,] "Barbadori"  "Medici"
##  [7,] "Bischeri"   "Guadagni"
##  [8,] "Bischeri"   "Peruzzi"
##  [9,] "Bischeri"   "Strozzi"
## [10,] "Castellani" "Peruzzi"
## [11,] "Castellani" "Strozzi"
## [12,] "Guadagni"   "Lamberteschi"
## [13,] "Guadagni"   "Tornabuoni"
## [14,] "Medici"     "Ridolfi"
## [15,] "Medici"     "Salviati"
## [16,] "Medici"     "Tornabuoni"
## [17,] "Pazzi"      "Salviati"
## [18,] "Peruzzi"    "Strozzi"
## [19,] "Ridolfi"    "Strozzi"
## [20,] "Ridolfi"    "Tornabuoni"
```

```
as_adjacency_matrix(medici)
```

```
## 16 x 16 sparse Matrix of class "dgCMatrix"
##   [[ suppressing 16 column names 'Acciaiuoli', 'Albizzi', 'Barbadori' ... ]]
##
## Acciaiuoli   . . . . . . . 1 . . . . . . .
## Albizzi      . . . . . 1 1 . 1 . . . . . .
## Barbadori    . . . . 1 . . . 1 . . . . . .
## Bischeri     . . . . . . 1 . . . 1 . . . 1 .
## Castellani   . . 1 . . . . . . 1 . . . 1 .
## Ginori       . 1 . . . . . . . . . . . . . .
## Guadagni     . 1 . 1 . . . 1 . . . . . . . 1
## Lamberteschi . . . . . . 1 . . . . . . . . .
## Medici       1 1 1 . . . . . . . . . 1 1 . 1
## Pazzi        . . . . . . . . . . . . . 1 . .
## Peruzzi      . . . 1 1 . . . . . . . . 1 . .
## Pucci        . . . . . . . . . . . . . . . .
## Ridolfi      . . . . . . . . 1 . . . . . 1 1
## Salviati     . . . . . . . . 1 1 . . . . . .
## Strozzi      . . . 1 1 . . . . . 1 . 1 . . .
## Tornabuoni   . . . . . . 1 . 1 . . . 1 . . .
```

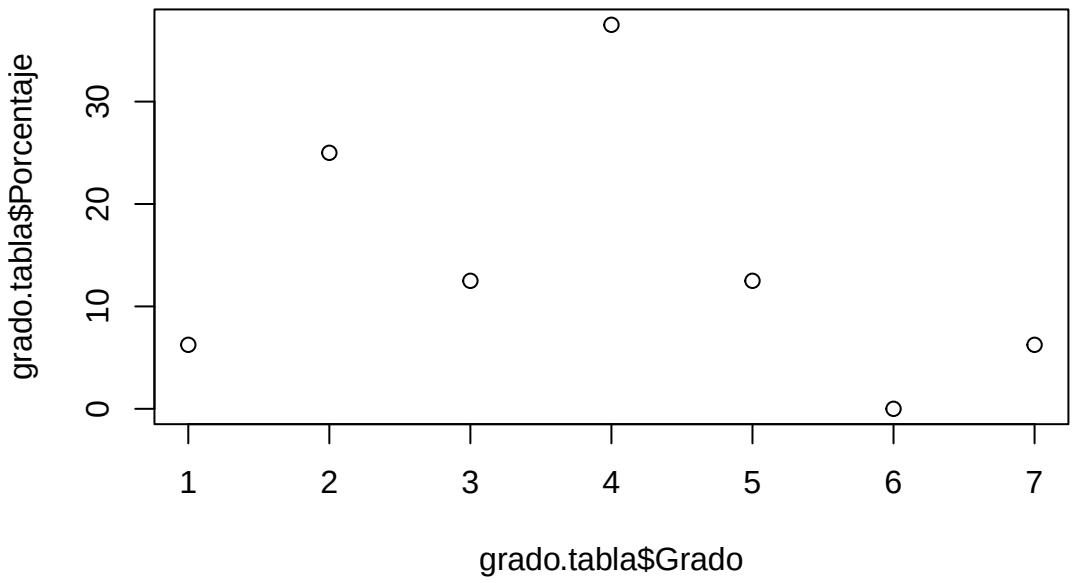
```
plot(medici)
```



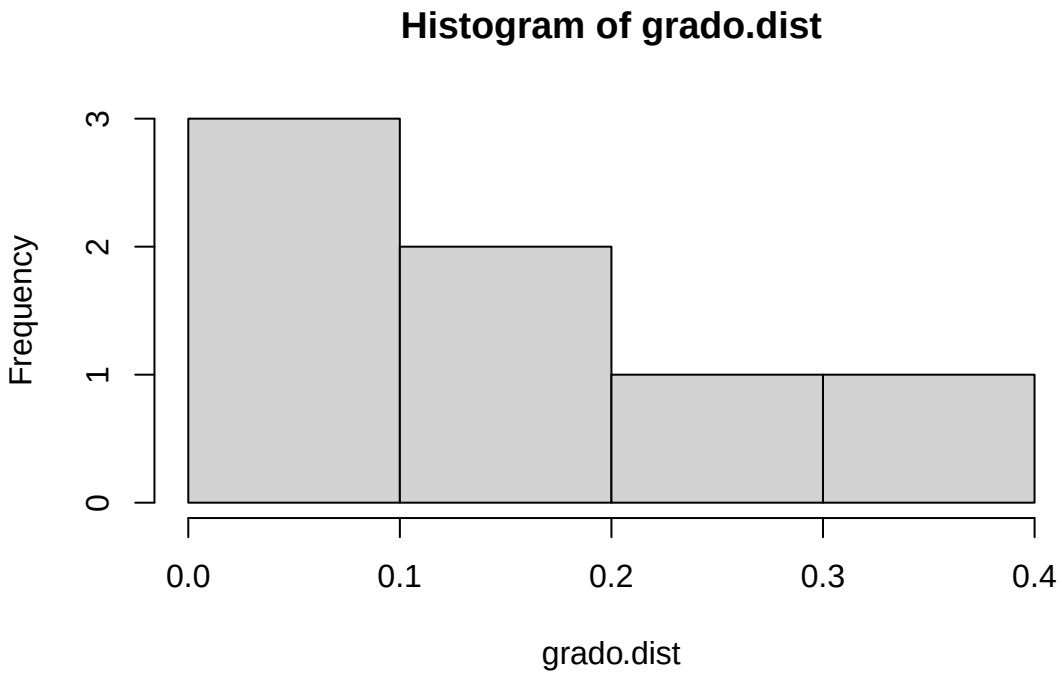

```
# distribucion de grado
grado.dist <- degree_distribution(medici)
grado.tabla <- matrix(c(seq(0:6),100*grado.dist),byrow=F,ncol=2)
grado.tabla <- as.data.frame(grado.tabla)
colnames(grado.tabla) <- c("Grado","Porcentaje")
grado.tabla
```

```
##   Grado Porcentaje
## 1     1       6.25
## 2     2      25.00
## 3     3      12.50
## 4     4      37.50
## 5     5      12.50
## 6     6       0.00
## 7     7       6.25
```

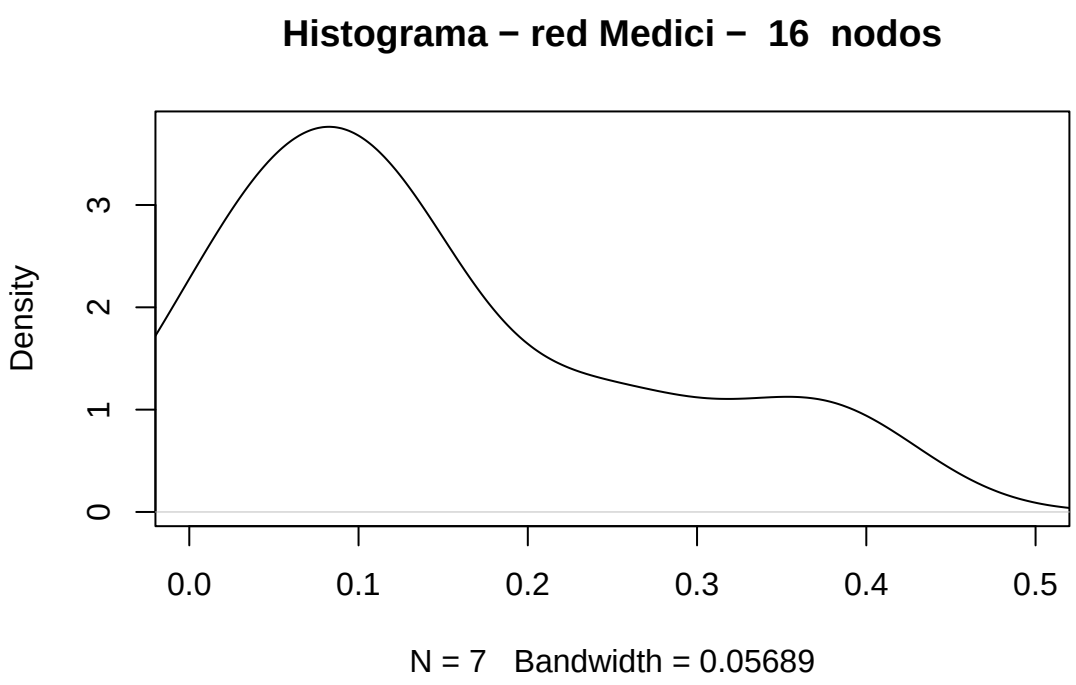
```
plot(grado.tabla$Grado,grado.tabla$Porcentaje)
```




```
hist(grado.dist)
```




```
t <- paste("Histograma - red Medici - ",length(V(medici))," nodos")
plot(density(grado.dist), main = t, xlim=c(0,0.5))
```




```
# componente gigante
componentes <- clusters(medici)
componentes
```

```
## $membership
##   Acciaiuoli      Albizzi    Barbadori     Bischeri   Castellani       Ginori
##            1            1            1            1            1            1
##     Guadagni Lamberteschi       Medici        Pazzi      Peruzzi        Pucci
##            1            1            1            1            1            2
##      Ridolfi     Salviati      Strozzi   Tornabuoni
##            1            1            1            1
##
## $csize
## [1] 15  1
##
## $no
## [1] 2
```

```
g <- which.max(componentes$csize) # identificamos el gigante
red <- induced.subgraph(medici, which(componentes$membership == g)) # nos quedamos con el componente gi
V(medici)
```

```
## + 16/16 vertices, named, from f55acc1:
##  [1] Acciaiuoli   Albizzi      Barbadori    Bischeri     Castellani
##  [6] Ginori       Guadagni     Lamberteschi Medici       Pazzi
## [11] Peruzzi      Pucci        Ridolfi      Salviati     Strozzi
## [16] Tornabuoni
```

```
V(red)
```

```
## + 15/15 vertices, named, from f5639df:
```

```
##  [1] Acciaiuoli   Albizzi      Barbadori    Bischeri     Castellani
##  [6] Ginori       Guadagni     Lamberteschi Medici       Pazzi
## [11] Peruzzi      Ridolfi      Salviati     Strozzi      Tornabuoni
```

```
# densidad
edge_density(red)
```

```
## [1] 0.1904762
```

```
# distancia media
distancia <- round(mean_distance(red),3)
distancia
```

```
## [1] 2.486
```

```
# geodesicas
m <- as_adjacency_matrix(red, sparse = F)
class(m)
```

```
## [1] "matrix" "array"
```

```
t(m)%*%m # matrix de dos grados
```

```
##              Acciaiuoli Albizzi Barbadori Bischeri Castellani Ginori Guadagni
## Acciaiuoli            1       1         1        0          0      0        0
## Albizzi               1       3         1        1          0      0        0
## Barbadori             1       1         2        0          0      0        0
## Bischeri              0       1         0        3          2      0        0
## Castellani            0       0         0        2          3      0        0
## Ginori                0       0         0        0          0      1        1
## Guadagni              0       0         0        0          0      1        4
## Lamberteschi          0       1         0        1          0      0        0
## Medici                0       0         0        0          1      1        2
## Pazzi                 0       0         0        0          0      0        0
## Peruzzi               0       0         1        1          1      0        1
## Ridolfi               1       1         1        1          1      0        1
## Salviati              1       1         1        0          0      0        0
## Strozzi               0       0         1        1          1      0        1
## Tornabuoni            1       2         1        1          0      0        0
##              Lamberteschi Medici Pazzi Peruzzi Ridolfi Salviati Strozzi
## Acciaiuoli              0      0     0       0       1        1       0
## Albizzi                 1      0     0       0       1        1       0
## Barbadori               0      0     0       1       1        1       1
## Bischeri                1      0     0       1       1        0       1
## Castellani              0      1     0       1       1        0       1
## Ginori                  0      1     0       0       0        0       0
## Guadagni                0      2     0       1       1        0       1
## Lamberteschi            1      0     0       0       0        0       0
## Medici                  0      6     1       0       1        0       1
## Pazzi                   0      1     1       0       0        0       0
## Peruzzi                 0      0     0       3       1        0       2
## Ridolfi                 0      1     0       1       3        1       0
## Salviati                0      0     0       0       1        2       0
## Strozzi                 0      1     0       2       0        0       4
## Tornabuoni              1      1     0       0       1        1       1
##              Tornabuoni
## Acciaiuoli             1
```

```
## Albizzi               2
## Barbadori             1
## Bischeri              1
## Castellani            0
## Ginori                0
## Guadagni              0
## Lamberteschi          1
## Medici                1
## Pazzi                 0
## Peruzzi               0
## Ridolfi               1
## Salviati              1
## Strozzi               1
## Tornabuoni            3
```

```
distance_table(red)
```

```
## $res
## [1] 20 35 32 15  3
##
## $unconnected
## [1] 0
```

```
shortest_paths(red,"Peruzzi","Pazzi")
```

```
## $vpath
## $vpath[[1]]
## + 6/15 vertices, named, from f5639df:
## [1] Peruzzi    Castellani Barbadori  Medici     Salviati   Pazzi
##
##
## $epath
## NULL
##
## $predecessors
## NULL
##
## $inbound_edges
## NULL
```

```
all_shortest_paths(red,"Peruzzi")
```

```
## $res
## $res[[1]]
## + 5/15 vertices, named, from f5639df:
## [1] Peruzzi    Castellani Barbadori  Medici     Acciaiuoli
##
## $res[[2]]
## + 5/15 vertices, named, from f5639df:
## [1] Peruzzi    Strozzi    Ridolfi    Medici     Acciaiuoli
##
## $res[[3]]
## + 4/15 vertices, named, from f5639df:
## [1] Peruzzi  Bischeri Guadagni Albizzi
##
## $res[[4]]
```

```
## + 3/15 vertices, named, from f5639df:
## [1] Peruzzi    Castellani Barbadori
##
## $res[[5]]
## + 2/15 vertices, named, from f5639df:
## [1] Peruzzi  Bischeri
##
## $res[[6]]
## + 2/15 vertices, named, from f5639df:
## [1] Peruzzi    Castellani
##
## $res[[7]]
## + 5/15 vertices, named, from f5639df:
## [1] Peruzzi  Bischeri Guadagni Albizzi  Ginori
##
## $res[[8]]
## + 3/15 vertices, named, from f5639df:
## [1] Peruzzi  Bischeri Guadagni
##
## $res[[9]]
## + 4/15 vertices, named, from f5639df:
## [1] Peruzzi      Bischeri     Guadagni     Lamberteschi
##
## $res[[10]]
## + 4/15 vertices, named, from f5639df:
## [1] Peruzzi Strozzi Ridolfi Medici
##
## $res[[11]]
## + 4/15 vertices, named, from f5639df:
## [1] Peruzzi    Castellani Barbadori  Medici
##
## $res[[12]]
## + 6/15 vertices, named, from f5639df:
## [1] Peruzzi  Strozzi  Ridolfi  Medici   Salviati Pazzi
##
## $res[[13]]
## + 6/15 vertices, named, from f5639df:
## [1] Peruzzi    Castellani Barbadori  Medici     Salviati   Pazzi
##
## $res[[14]]
## + 1/15 vertex, named, from f5639df:
## [1] Peruzzi
##
## $res[[15]]
## + 3/15 vertices, named, from f5639df:
## [1] Peruzzi Strozzi Ridolfi
##
## $res[[16]]
## + 5/15 vertices, named, from f5639df:
## [1] Peruzzi    Castellani Barbadori  Medici     Salviati
##
## $res[[17]]
## + 5/15 vertices, named, from f5639df:
## [1] Peruzzi  Strozzi  Ridolfi  Medici   Salviati
```

```
##
## $res[[18]]
## + 2/15 vertices, named, from f5639df:
## [1] Peruzzi Strozzi
##
## $res[[19]]
## + 4/15 vertices, named, from f5639df:
## [1] Peruzzi    Strozzi    Ridolfi    Tornabuoni
##
## $res[[20]]
## + 4/15 vertices, named, from f5639df:
## [1] Peruzzi    Bischeri   Guadagni   Tornabuoni
##
##
## $nrgeo
##  [1] 2 1 1 1 1 1 1 1 2 2 1 1 2 1 2
```

```
# conteo de triadas
triad_census(red) # entrega un censo según clasificación: https://igraph.org/r/doc/triad_census.html
```

```
## Warning in triad_census(red): At core/misc/motifs.c:1165 : Triad census called
## on an undirected graph.
```

```
##  [1] 239   0 175   0   0   0   0   0   0   0  38   0   0   0   0   3
```