

Name: Yash Kesharwani
Class: MSc CS Part I

Academic Year: 2021-2022
Roll no.: 3

Subject: Social Network Analysis

Index

Sr. No.	Title	Page No.
1	Write a program to compute the following for a given a network: number of edges, number of nodes, degree of node, node with lowest degree, the adjacency list, matrix of the graph.	2
2	Perform following tasks: View data collection forms and/or import onemode/two-mode datasets, Basic Networks matrices transformations	11
3	Compute the following node level measures: Density; Degree; Reciprocity; Transitivity; Centralization; Clustering.	15
4	For a given network find the following: Length of the shortest path from a given node to another node; the density of the graph; Draw egocentric network of node G with chosen configuration parameters.	22
5	Write a program to distinguish between a network as a matrix, a network as an edge list, and a network as a sociogram (or “network graph”) using 3 distinct networks representatives of each.	29
6	Write a program to exhibit structural equivalence, automatic equivalence, and regular equivalence from a network.	33
7	Perform SVD analysis of a network.	38

Practical 1

Aim: Write a program to compute the following for a given a network:

- ◆ number of edges
- ◆ number of nodes
- ◆ degree of node
- ◆ node with lowest degree
- ◆ the adjacency list
- ◆ matrix of the graph.

Software(s) used:

- ◆ R ver. 4.1.3
- ◆ RStudio ver. 2022.02.0+433

External packages required:

- ◆ igraph

Description:

- **The igraphpackage:** igraph is a library and R package for network analysis.
The main goals of the igraph library is to provide a set of data types and functions for:
 - ◆ pain-free implementation of graph algorithms,
 - ◆ fast handling of large graphs, with millions of vertices and edges, allowing rapid prototyping via high level languages like R.

- `library()` `library()` loads and attach add-on packages.
- `graph.formula()`
 - ◆ Creating (small) graphs via a simple interface
 - ◆ This function is useful if you want to create a small (named) graph quickly, it works for both directed and undirected graphs.
- `plot()` Use to plot any graph.
- `ecount()` Returns the count of number of edges in graph
- `vcount()` Returns the count of number of vertices in graph
- `E()`:
 - ◆ Edges of a graph
 - ◆ An edge sequence is a vector containing numeric edge ids, with a special class attribute that allows custom operations: selecting subsets of edges based on attributes, or graph structure, creating the intersection, union of edges, etc.
- `V()`:
 - ◆ Vertices of a graph
 - ◆ Create a vertex sequence (vs) containing all vertices of a graph.
- `degree()`
 - ◆ Degree and degree distribution of the vertices
 - ◆ The degree of a vertex is its most basic structural property, the number of its adjacent edges.
 - ◆ Mode-Character string, “out” for out-degree, “in” for in-degree or “total” for the sum of the two. For undirected graphs this argument is ignored. “all” is a synonym of “total”.

- `max()` and `min()`
 - ◆ Maxima and Minima
 - ◆ Returns the (regular or parallel) maxima and minima of the input values.
- `get.adjacency()` Convert a graph to an adjacency matrix
- `get.adjlist()`
 - ◆ Adjacency lists
 - ◆ Create adjacency lists from a graph, either for adjacent edges or for neighboring vertices

Source Code:

```
library(igraph)

u_graph <- graph.formula(A - B, A - C, A - D, B - C, B - F, C -
D, C - E, C - F, D - E, E - F, F - G, G - H)
d_graph <- graph.formula(A <+ B, A <+ D, A -+ C, B -+ C, B -+
E, B -+ F, C -+ D, C -+ F, D -+ E)

e_count(u_graph)
e_count(d_graph)

v_count(u_graph)
v_count(d_graph)

E(u_graph)
E(d_graph)

V(u_graph)
V(d_graph)
```

```
degree(u_graph)
degree(u_graph, mode = "in")
degree(u_graph, mode = "out")
```

```
degree(d_graph)
degree(d_graph, mode = "in")
degree(d_graph, mode = "out")
```

```
V(u_graph)$name[degree(u_graph) <= min(degree(u_graph))]  
V(d_graph)$name[degree(d_graph, mode = "in") <=  
min(degree(d_graph, mode = "in"))]  
V(d_graph)$name[degree(d_graph, mode = "out") <=  
min(degree(d_graph, mode = "out"))]
```

```
get.adjacency(u_graph)  
get.adjacency(d_graph)
```

```
get.adjlist(u_graph)  
get.adjlist(d_graph)
```

Output:

```
> library(igraph)
>
> u_graph <- graph.formula(A - B, A - C, A - D, B - C, B - F, C - D, C - E, C - F, D - E, E - F, F - G, G
- H)
> d_graph <- graph.formula(A ++ B, A ++ D, A -- C, B -- C, B -- E, B -- F, C -- D, C -- F, D -- E)
>
> ecount(d_graph)
[1] 11
> ecount(d_graph)
[1] 11
>
> vcount(u_graph)
[1] 8
> vcount(d_graph)
[1] 6
>
> E(u_graph)
+ 12/12 edges from 535b776 (vertex names):
[1] A--B A--C A--D B--C B--F C--D C--E D--E F--E F--G G--H
> E(d_graph)
+ 11/11 edges from 5365b0b (vertex names):
[1] A->B A->D A->C B->A B->C B->E B->F D->A D->E C->D C->F
```

```

> V(u_graph)
+ 8/8 vertices, named, from 535b776:
[1] A B C D F E G H
> V(d_graph)
+ 6/6 vertices, named, from 5365b0b:
[1] A B D C E F
>
> degree(u_graph)
A B C D F E G H
3 3 5 3 4 3 2 1
> degree(u_graph, mode = "in")
A B C D F E G H
3 3 5 3 4 3 2 1
> degree(u_graph, mode = "out")
A B C D F E G H
3 3 5 3 4 3 2 1
>
> degree(d_graph)
A B D C E F
5 5 4 4 2 2

```

```

> degree(d_graph, mode = "in")
A B D C E F
2 1 2 2 2 2
> degree(d_graph, mode = "out")
A B D C E F
3 4 2 2 0 0
>
> V(u_graph)$name[degree(u_graph) == min(degree(u_graph))]
[1] "H"
> V(d_graph)$name[degree(d_graph, mode = "in") == min(degree(d_graph, mode = "in"))]
[1] "B"
> V(d_graph)$name[degree(d_graph, mode = "out") == min(degree(d_graph, mode = "out"))]
[1] "E" "F"
>

```

```

> get.adjacency(u_graph)
8 x 8 sparse Matrix of class "dgCMatrix"
  A B C D F E G H
A . 1 1 1 . . . .
B 1 . 1 . 1 . . .
C 1 1 . 1 1 1 . .
D 1 . 1 . . 1 . .
F . 1 1 . . 1 1 .
E . . 1 1 1 . . .
G . . . . 1 . . 1
H . . . . . . 1 .
> get.adjacency(d_graph)
6 x 6 sparse Matrix of class "dgCMatrix"
  A B D C E F
A . 1 1 1 . .
B 1 . . 1 1 1
D 1 . . . 1 .
C . . 1 . . 1
E . . . . . .
F . . . . . .
>

```

```
> get.adjlist(u_graph)
$A
+ 3/8 vertices, named, from 535b776:
[1] B C D

$B
+ 3/8 vertices, named, from 535b776:
[1] A C F

$C
+ 5/8 vertices, named, from 535b776:
[1] A B D F E

$D
+ 3/8 vertices, named, from 535b776:
[1] A C E

$F
+ 4/8 vertices, named, from 535b776:
[1] B C E G
```



```
$E
+ 3/8 vertices, named, from 535b776:
[1] C D F

$G
+ 2/8 vertices, named, from 535b776:
[1] F H

$H
+ 1/8 vertex, named, from 535b776:
[1] G

> get.adjlist(d_graph)
$A
+ 5/6 vertices, named, from 5365b0b:
[1] B B D D C

$B
+ 5/6 vertices, named, from 5365b0b:
[1] A A C E F
```

```
$D  
+ 4/6 vertices, named, from 5365b0b:  
[1] A A C E
```

```
$C  
+ 4/6 vertices, named, from 5365b0b:  
[1] A B D F
```

```
$E  
+ 2/6 vertices, named, from 5365b0b:  
[1] B D
```

```
$F  
+ 2/6 vertices, named, from 5365b0b:  
[1] B C
```

Practical 2

Aim: Perform following tasks:

- ◆ View data collection forms and/or import onemode/two-mode datasets.
- ◆ Basic Networks matrices transformations.

Software(s) used:

- ◆ R ver. 4.1.3
- ◆ RStudio ver. 2022.02.0+433

External packages required:

- ◆ igraph

Description:

- **The igraphpackage:** igraph is a library and R package for network analysis.
The main goals of the igraph library is to provide a set of data types and functions for:
 - ◆ pain-free implementation of graph algorithms,
 - ◆ fast handling of large graphs, with millions of vertices and edges, allowing rapid prototyping via high level languages like R.
- `getwd()` Used to get the absolute filepath of the current R session.
- `require()` `library()` and `require()` load and attach add-on packages.

- `read.csv()` Reads a file in table format and creates a data frame from it, with cases corresponding to lines and variables to fields in the file.
- `head()` Returns the first part of a vector, matrix, table, data frame or function. Since `head()` and `tail()` are generic functions, they may also have been extended to other classes.
- `graph.data.frame()` This function creates an igraph graph from one or two data frames containing the (symbolic) edge list and edge/vertex attributes.
- `get.adjacency()` Sometimes it is useful to work with a standard representation of a graph, like an adjacency matrix.
- `plot()` Draw a scatter plot with decorations such as axes and titles in the active graphics window.

Source Code:

```
require("igraph")

edges <- read.csv("C:/Temp/input_files/edges.csv")
nodes <- read.csv("C:/Temp/input_files/nodes.csv")

head(edges)
head(nodes)

dir_graph <- graph.data.frame(d = edges, vertices = nodes,
directed = T)
get.adjacency(dir_graph)

plot(dir_graph)
```

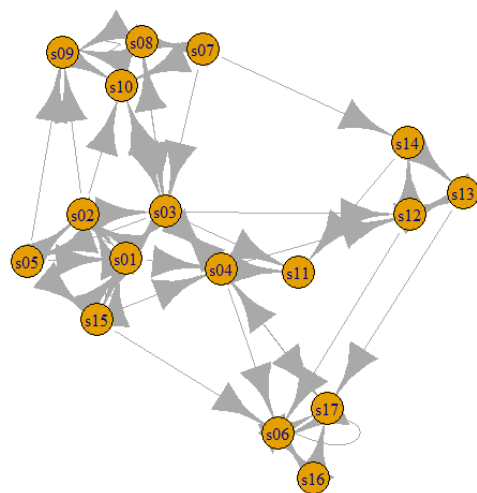
Output:

```
> require("igraph")
>
> edges <- read.csv("C:/Temp/input_files/edges.csv")
> nodes <- read.csv("C:/Temp/input_files/nodes.csv")
>
> head(edges)
  from to weight  type
1 s01 s02    10 hyperlink
2 s01 s02    12 hyperlink
3 s01 s03    22 hyperlink
4 s01 s04    21 hyperlink
5 s04 s11    22  mention
6 s05 s15    21  mention
> head(nodes)
  id          media media.type type.label audience.size
1 s01          NY Times          1 Newspaper          20
2 s02 Washington Post          1 Newspaper          25
3 s03 Wall Street Journal          1 Newspaper          30
4 s04          USA Today          1 Newspaper          32
5 s05          LA Times          1 Newspaper          20
6 s06 New York Post          1 Newspaper          50
>
```

```
> dir_graph <- graph.data.frame(d = edges, vertices = nodes, directed = T)
> get.adjacency(dir_graph)
17 x 17 sparse Matrix of class "dgCMatrix"
[[ suppressing 17 column names 's01', 's02', 's03' ... ]]

s01 . 2 1 1 . . . . . . . . . 1 . .
s02 1 . 1 . . . . . 1 1 . . . . .
s03 1 . . 1 1 . . 1 . 1 1 1 . . . .
s04 . . 1 . . 1 . . . . 1 1 . . . 1
s05 1 1 . . . . . . 1 . . . . . 1 .
s06 . . . . . 1 . . . . . . . . 1 1
s07 . . 1 . . . . 1 . 1 . . . 1 . .
s08 . . 1 . . . 1 . 2 . . . . . .
s09 . . . . . . . . . 1 . . . . .
s10 . . 1 . . . . . . . . . . . .
s11 . . . . . . . . . . . . . . .
s12 . . . . . 1 . . . . . 1 1 . . .
s13 . . . . . . . . . . . 1 . . . 1
s14 . . . . . . . . . . 1 . 1 . . .
s15 2 . . 1 . 1 . . . . . . . . .
s16 . . . . . 1 . . . . . . . . 1
s17 . . . 1 . . . . . . . . . . .
```

```
>  
> plot(dir_graph)  
> |
```



Practical 3

Aim: Compute the following node level measures:

- ◆ Density
- ◆ Degree
- ◆ Reciprocity
- ◆ Transitivity
- ◆ Centralization
- ◆ Clustering.

Software(s) used:

- ◆ R ver. 4.1.3
- ◆ RStudio ver. 2022.02.0+433

External packages required:

- ◆ igraph

Description:

- **The igraphpackage:** igraph is a library and R package for network analysis.
The main goals of the igraph library is to provide a set of data types and functions for:
 - ◆ pain-free implementation of graph algorithms,
 - ◆ fast handling of large graphs, with millions of vertices and edges, allowing rapid prototyping via high level languages like R.

- `library()` `library()` and `require()` load and attach add-on packages.
- `graph.famous(:)` Create an igraph graph from a list of edges, or a notable graph.
- `ecount()` Returns the count of number of edges in graph
- `vcount()` Returns the count of number of vertices in graph
- `graph.formula(:)` This function is useful if you want to create a small (named) graph quickly, it works for both directed and undirected graphs.
- `plot()` Draw a scatter plot with decorations such as axes and titles in the active graphics window.
- `reciprocity(:)` Calculates the reciprocity of a directed graph.
- `dyad.census()` Classify dyads in a directed graphs. The relationship between each pair of vertices is measured. It can be in three states: mutual, asymmetric or non-existent.
- `adjacent.triangles()` Count how many triangles a vertex is part of, in a graph, or just list the triangles of a graph.
- `transitivity(:)` Transitivity measures the probability that the adjacent vertices of a vertex are connected. This is sometimes also called the clustering coefficient.
- `degree()` The degree of a vertex is its most basic structural property, the number of its adjacent edges.

- `barabasi.game()` The BA-model is a very simple stochastic algorithm for building a graph.
- `watts.strogatz.game()` Generate a graph according to the Watts-Strogatz network model.
- `graph.union()` The union of two or more graphs are created. The graphs may have identical or overlapping vertex sets.
- `simplify()` Simple graphs are graphs which do not contain loop and multiple edges.

Source Code:

```
library("igraph")

kite <- graph.famous("Krackhardt_Kite")
vcount(kite)
ecount(kite)
ecount(kite) / (vcount(kite) * (vcount(kite) - 1) / 2)

dir_graph <- graph.formula(A <+ B, A <+ D, A -+ C, B -+ C, B -+
E, B -+ F, C -+ D, C -+ F, D -+ E)
plot(dir_graph)
reciprocity(dir_graph)
dyad.census(dir_graph)
mutual <- dyad.census(dir_graph)$mut
```

```
mutual / (ecount(dir_graph))
```

```
atri <- adjacent.triangles(kite)
```

```
plot(kite, vertex.label = atri)
```

```
transitivity(kite, type = "local")
```

```
adjacent.triangles(kite) / (degree(kite) * (degree(kite) - 1) /  
2)
```

```
graph_2 <- barabasi.game(50, p = 2, directed = F)
```

```
graph_1 <- watts.strogatz.game(1, size = 100, nei = 5, p =  
0.05)
```

```
graph <- graph.union(graph_1, graph_2)
```

```
graph <- simplify(graph)
```

```
plot(graph)
```

Output:

```
> library("igraph")  
>  
> kite <- graph.famous("Krackhardt_Kite")  
> vcount(kite)  
[1] 10  
> ecount(kite)  
[1] 18  
> ecount(kite) / (vcount(kite) * (vcount(kite) - 1) / 2)  
[1] 0.4  
>  
> dir_graph <- graph.formula(A ++ B, A ++ D, A -- C, B -- C, B -- E, B -- F, C -- D, C -- F, D -- E)  
> plot(dir_graph)  
> reciprocity(dir_graph)  
[1] 0.3636364
```

```

> reciprocity(dir_graph)
[1] 0.3636364
> dyad.census(dir_graph)
$mut
[1] 2

$asym
[1] 7

>null
[1] 6

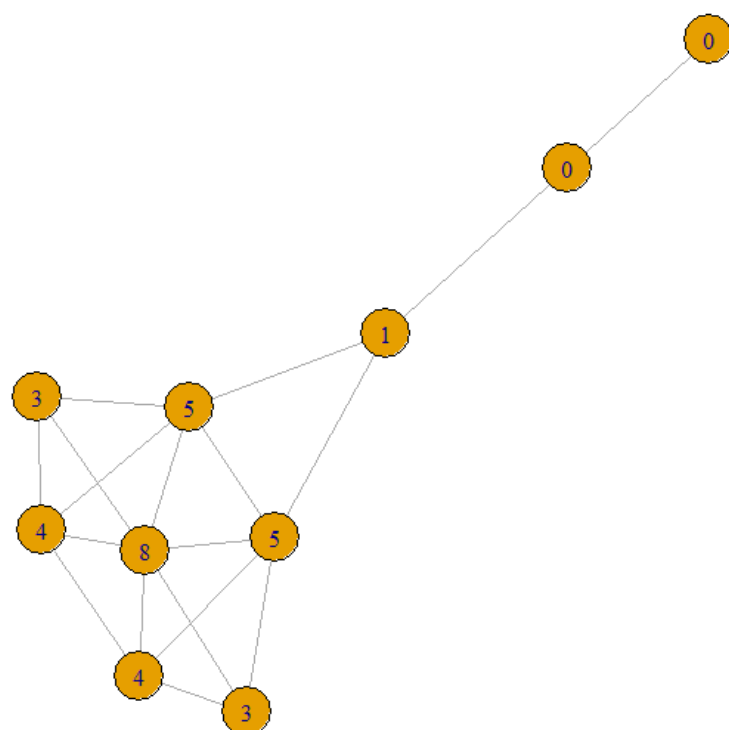
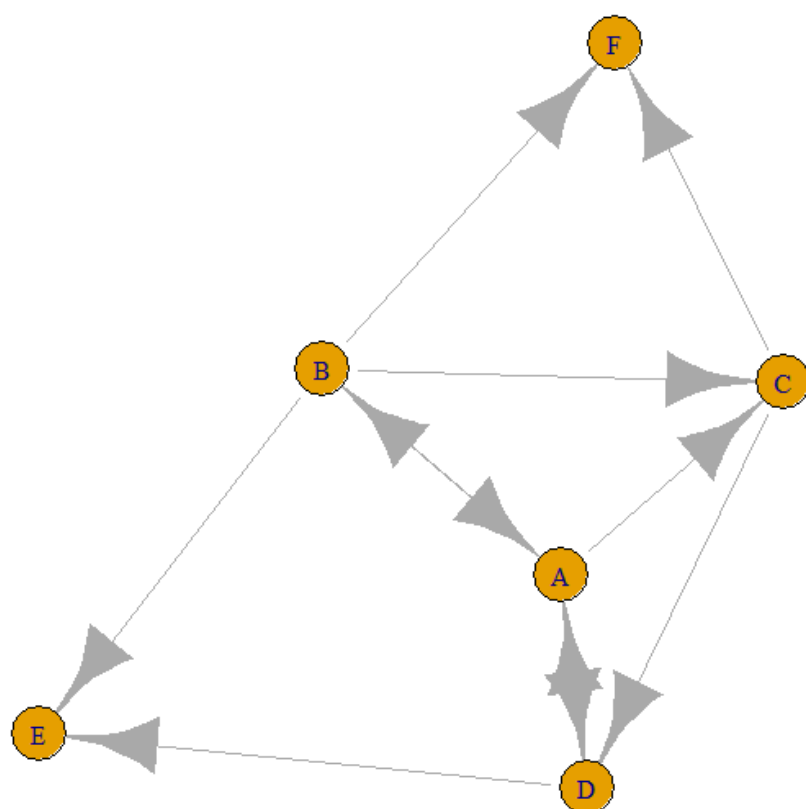
> mutual <- dyad.census(dir_graph)$mut
> mutual / (ecount(dir_graph))
[1] 0.1818182
>
> atri <- adjacent.triangles(kite)
> plot(kite, vertex.label = atri)
> transitivity(kite, type = "local")
[1] 0.6666667 0.6666667 1.0000000 0.5333333 1.0000000 0.5000000 0.5000000 0.3333333 0.0000000
[10]      NaN

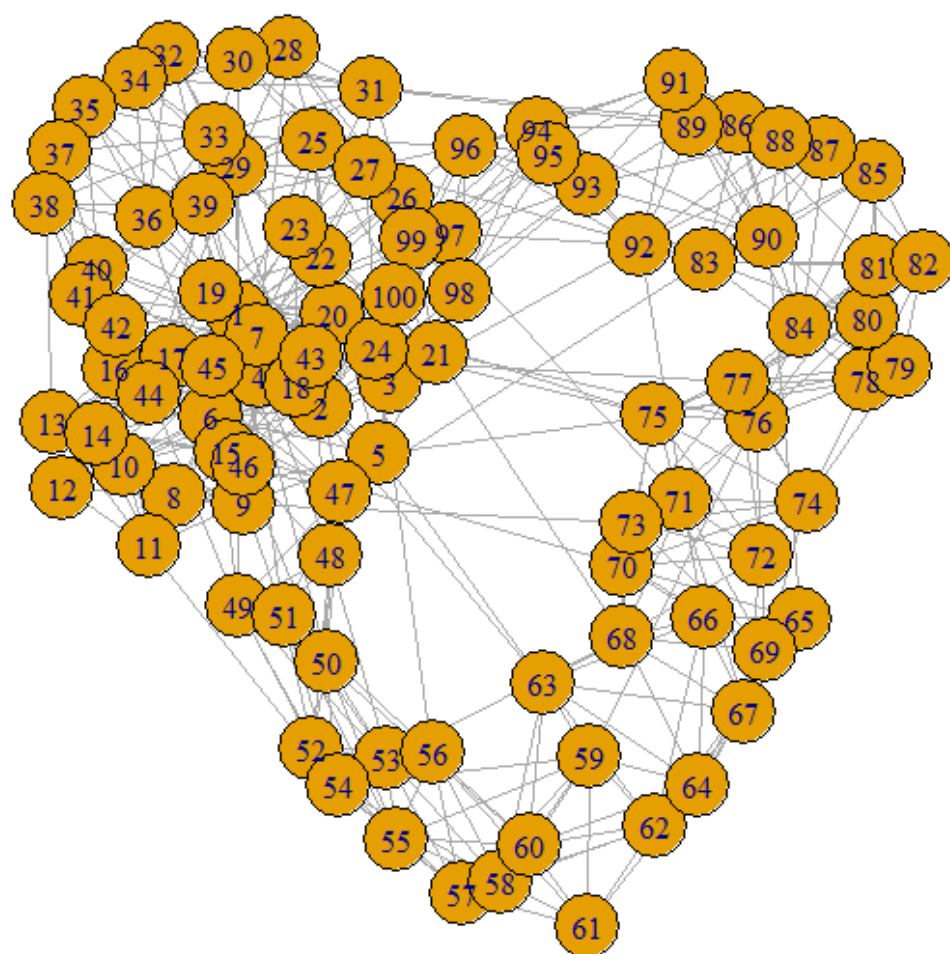
```

```

> adjacent.triangles(kite) / (degree(kite) * (degree(kite) - 1) / 2)
[1] 0.6666667 0.6666667 1.0000000 0.5333333 1.0000000 0.5000000 0.5000000 0.3333333 0.0000000
[10]      NaN
>
> graph_2 <- barabasi.game(50, p = 2, directed = F)
> graph_1 <- watts.strogatz.game(1, size = 100, nei = 5, p = 0.05)
> graph <- graph.union(graph_1, graph_2)
> graph <- simplify(graph)
> plot(graph)

```





Practical 4

Aim: For a given network find the following:

- ◆ Length of the shortest path from a given node to another node
- ◆ The density of the graph
- ◆ Draw egocentric network of node G with chosen configuration parameters.

Software(s) used:

- ◆ R ver. 4.1.3
- ◆ RStudio ver. 2022.02.0+433

External packages required:

- ◆ igraph

Description:

- **The igraph package:** igraph is a library and R package for network analysis.
The main goals of the igraph library is to provide a set of data types and functions for:
 - ◆ pain-free implementation of graph algorithms,
 - ◆ fast handling of large graphs, with millions of vertices and edges, allowing rapid prototyping via high level languages like R.
- `library()` `library()` and `require()` load and attach add-on packages.

- `as.matrix()``matrix()` creates a matrix from the given set of values. `as.matrix()` attempts to turn its argument into a matrix.
- `read.table()` Reads a file in table format and creates a data frame from it, with cases corresponding to lines and variables to fields in the file.
- `colnames()` and `rownames()` Retrieve or set the row or column names of a matrix-like object.
- `is.na()` The generic function `is.na()` indicates which elements are missing.
- `graph.adjacency()``graph_from_adjacency_matrix()` is a flexible function for creating igraph graphs from adjacency matrices.
- `plot()` Draw a scatter plot with decorations such as axes and titles in the active graphics window.
- `shortest.paths()``shortest_paths()` calculates one shortest path (the path itself, and not just its length) from or to the given vertex.
- `print()` `print()` prints its argument and returns it invisibly.
- `graph.formula()` This function is useful if you want to create a small (named) graph quickly, it works for both directed and undirected graphs.
- `graph.density()` The density of a graph is the ratio of the number of edges and the number of possible edges.

- `simplify()` Simple graphs are graphs which do not contain loop and multiple edges.

Source Code:

```
library(igraph)

matt <- as.matrix(read.table(text=
                                "node R S T U
                                R 7 5 0 0
                                S 7 0 0 2
                                T 0 6 0 0
                                U 4 0 1 0", header=T))

nms <- matt[, 1]
matt <- matt[, -1]
colnames(matt) <- rownames(matt) <- nms
matt[is.na(matt)] <- 0
g <- graph.adjacency(matt,weighted=TRUE)
plot(g)

s.paths <- shortest.paths(g, algorithm = "dijkstra")
print(s.paths)

shortest.paths(g, v="R", to="S")
plot(g, edge.label=E(g)$weight)
```



```
dg <- graph.formula(1-+2, 1-+3, 2<+3)
```

```
plot(dg)
```

```
graph.density(dg, loops=TRUE)
```

```
graph.density(simplify(dg), loops=FALSE)
```

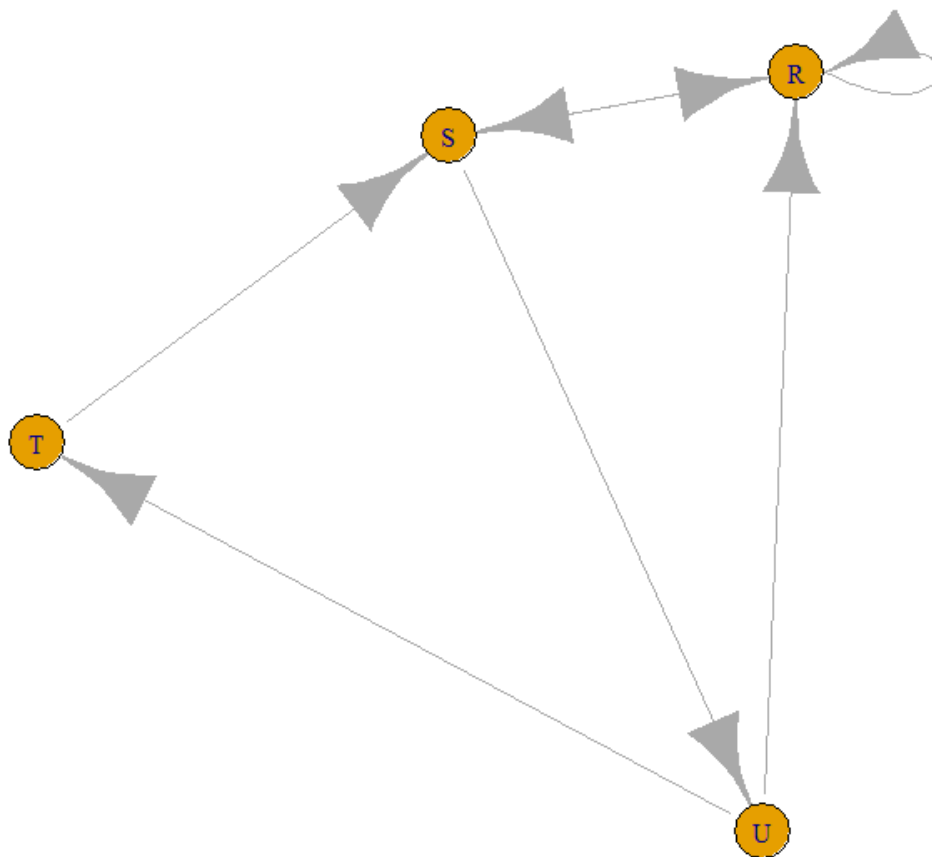
Output:

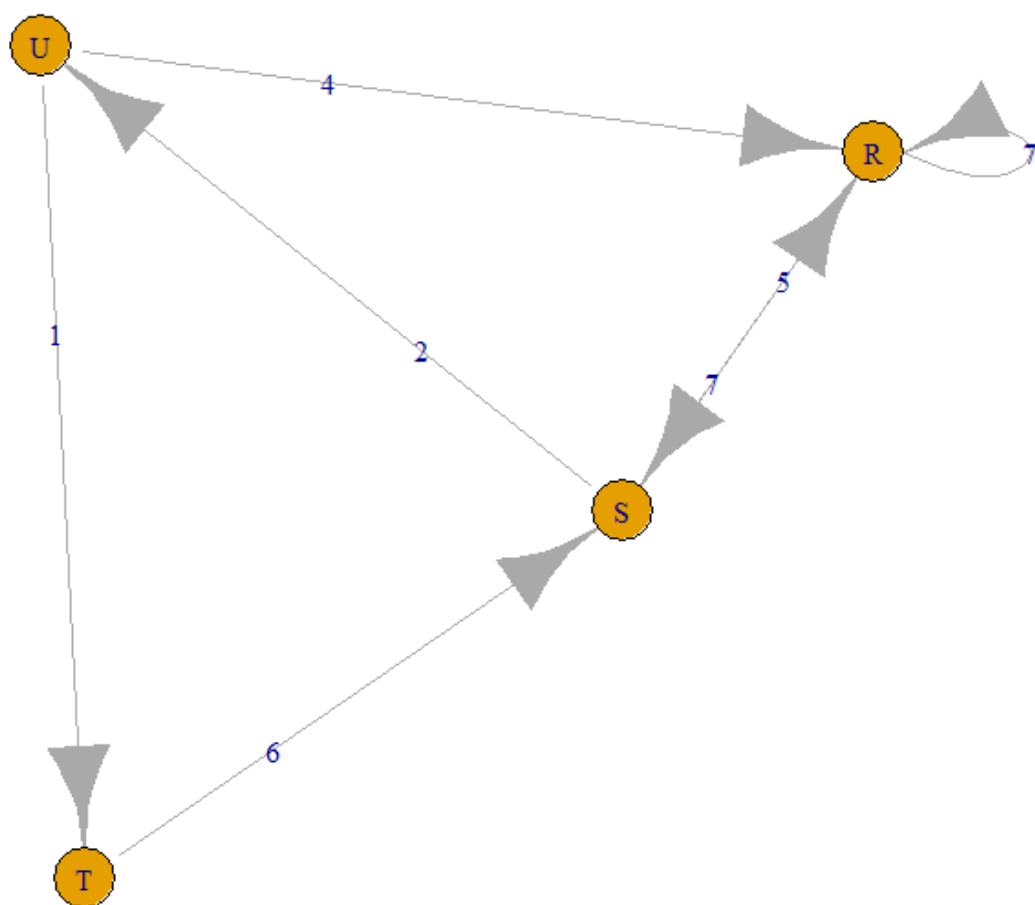
```
> library(igraph)
> matt <- as.matrix(read.table(text=
+           "node R S T U
+           R 7 5 0 0
+           S 7 0 0 2
+           T 0 6 0 0
+           U 4 0 1 0", header=T))
> nms <- matt[, 1]
> matt <- matt[, -1]
> colnames(matt) <- rownames(matt) <- nms
> matt[is.na(matt)] <- 0
> g <- graph.adjacency(matt,weighted=TRUE)
> plot(g)
>
> s.paths <- shortest.paths(g, algorithm = "dijkstra")
> print(s.paths)
  R S T U
R 0 5 5 4
S 5 0 3 2
T 5 3 0 1
U 4 2 1 0
>
```

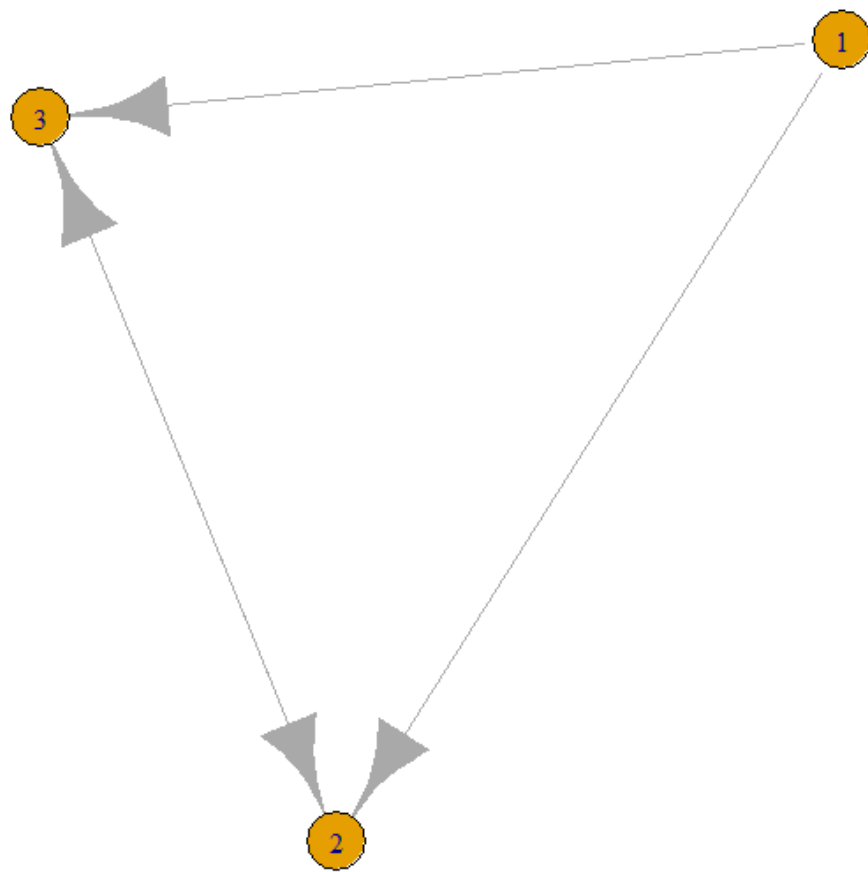
```

> shortest.paths(g, v="R", to="S")
  S
R 5
> plot(g, edge.label=E(g)$weight)
>
> dg <- graph.formula(1->2, 1->3, 2->3)
> plot(dg)
>
> graph.density(dg, loops=TRUE)
[1] 0.4444444
> graph.density(simplify(dg), loops=FALSE)
[1] 0.6666667
>

```







Practical 5

Aim: Write a program to distinguish between a network as a matrix, a network as an edge list and a network as a sociogram (or “network graph”) using 3 distinct networks representatives of each.

Software(s) used:

- ◆ R ver. 4.1.3
- ◆ RStudio ver. 2022.02.0+433

External packages required:

- ◆ igraph

Description:

- **The igraphpackage:** igraph is a library and R package for network analysis.
The main goals of the igraph library is to provide a set of data types and functions for:
 - ◆ pain-free implementation of graph algorithms,
 - ◆ fast handling of large graphs, with millions of vertices and edges, allowing rapid prototyping via high level languages like R.
- `library()` `library()` and `require()` load and attach add-on packages.

- `graph.formula()` This function is useful if you want to create a small (named) graph quickly, it works for both directed and undirected graphs.
- `plot()` Draw a scatter plot with decorations such as axes and titles in the active graphics window.
- `get.adjacency()` Sometimes it is useful to work with a standard representation of a graph, like an adjacency matrix.
- `E()` : An edge sequence is a vector containing numeric edge ids, with a special class attribute that allows custom operations: selecting subsets of edges based on attributes, or graph structure, creating the intersection, union of edges, etc.
- `get.edgelist()` Create adjacency lists from a graph, either for adjacent edges or for neighboring vertices.

Source Code:

```
library(igraph)
```

```
sociogram <- graph.formula(Andy<+Garth, Garth-+Bill, Bill-
                        +Elena, Elena<+Frank, Carol-
                        +Andy, Carol-+Elena, Carol+
                        +Dan, Carol<+Bill, Dan<+Andy,
                        Dan<+Bill)
```

```
plot(sociogram)
```

```
get.adjacency(sociogram)
```

```
E(sociogram)
```

```
get.edgelist(sociogram)
```

Output:

```
> library(igraph)
>
> sociogram <- graph.formula(Andy++Garth, Garth-->Bill, Bill-
+                               +Elena, Elena++Frank, Carol-->Andy, Carol-
+                               +Elena, Carol++Dan, Carol++Bill, Dan++Andy, Dan++Bill)
>
> plot(sociogram)
>
> get.adjacency(sociogram)
7 x 7 sparse Matrix of class "dgCMatrix"
      Andy Garth Bill Elena Frank Carol Dan
Andy    .    1    .    .    .    .    1
Garth   1    .    1    .    .    .    .
Bill    .    .    .    1    .    1    1
Elena   .    .    .    .    1    .    .
Frank   .    .    .    1    .    .    .
Carol   1    .    1    1    .    .    1
Dan     1    .    1    .    .    1    .
```

```
> E(sociogram)
+ 16/16 edges from f262a60 (vertex names):
[1] Andy →Garth Andy →Dan   Garth→Andy   Garth→Bill
[5] Bill →Elena Bill →Carol Bill →Dan   Elena→Frank
[9] Frank→Elena Carol→Andy   Carol→Bill   Carol→Elena
[13] Carol→Dan   Dan →Andy   Dan →Bill   Dan →Carol
> get.edgelist(sociogram)
$Andy
+ 5/16 edges from f262a60 (vertex names):
[1] Andy →Garth Andy →Dan   Garth→Andy   Carol→Andy
[5] Dan →Andy

$Garth
+ 3/16 edges from f262a60 (vertex names):
[1] Garth→Andy   Garth→Bill   Andy →Garth

$Bill
+ 6/16 edges from f262a60 (vertex names):
[1] Bill →Elena Bill →Carol Bill →Dan   Garth→Bill
[5] Carol→Bill   Dan →Bill
```

```

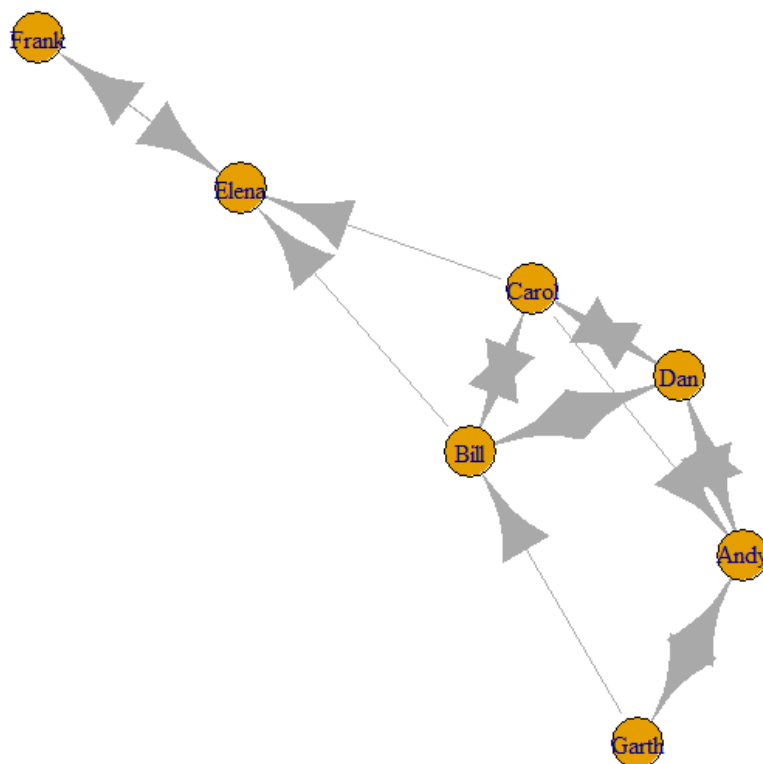
$Elena
+ 4/16 edges from f262a60 (vertex names):
[1] Elena→Frank Bill →Elena Frank→Elena Carol→Elena

$Frank
+ 2/16 edges from f262a60 (vertex names):
[1] Frank→Elena Elena→Frank

$Carol
+ 6/16 edges from f262a60 (vertex names):
[1] Carol→Andy Carol→Bill Carol→Elena Carol→Dan
[5] Bill →Carol Dan →Carol

$Dan
+ 6/16 edges from f262a60 (vertex names):
[1] Dan →Andy Dan →Bill Dan →Carol Andy →Dan
[5] Bill →Dan Carol→Dan

```



Practical 6

Aim: Write a program to exhibit structural equivalence, automatic equivalence, and regular equivalence from a network.

Software(s) used:

- ◆ R ver. 4.1.3
- ◆ RStudio ver. 2022.02.0+433

External packages required:

- ◆ igraph
- ◆ sna

Description:

- **The igraph package:** igraph is a library and R package for network analysis.
The main goals of the igraph library is to provide a set of data types and functions for:
 - ◆ pain-free implementation of graph algorithms,
 - ◆ fast handling of large graphs, with millions of vertices and edges, allowing rapid prototyping via high level languages like R.
- **The sna package:** sna is a package containing a range of tools for social network analysis. Supported functionality includes node and graph-level indices, structural distance and covariance methods, structural equivalence detection, p^* modeling, random graph generation, and 2D/3D network visualization (among other things).

- `library()` `library()` and `require()` load and attach add-on packages.
- `read.csv()` Reads a file in table format and creates a data frame from it, with cases corresponding to lines and variables to fields in the file.
- `equiv.clust()` `equiv.clust()` uses a definition of approximate equivalence (`equiv.fun()`) to form a hierarchical clustering of network positions.
- `plot()` Draw a scatter plot with decorations such as axes and titles in the active graphics window.
- `sedist()` `sedist()` uses the graphs indicated by `g` in the arguments to assess the extent to which each vertex is structurally equivalent.
- `cmdscale()` Classical multidimensional scaling (MDS) of a data matrix.
- `as.dist()` This function computes and returns the distance matrix computed by using the specified distance measure to compute the distances between the rows of a data matrix.
- `blockmodel()` Given a set of equivalence classes and one or more graphs, `blockmodel` will form a blockmodel of the input graph(s) based on the classes in question

Source Code:

```
library(sna)

library(igraph)

links2 <- read.csv("C:/Temp/input_files/edges1.csv", header=T,
row.names=1)

eq<-equiv.clust(links2)

plot(eq)


g.se<-sedist(links2)


plot(cmdscale(as.dist(g.se)))

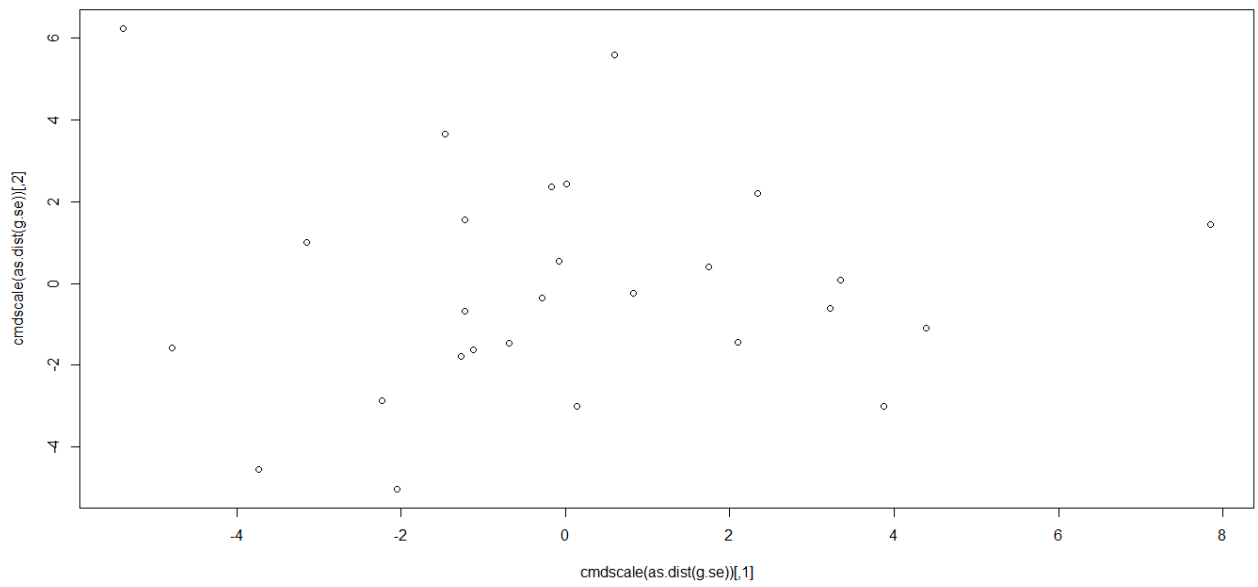
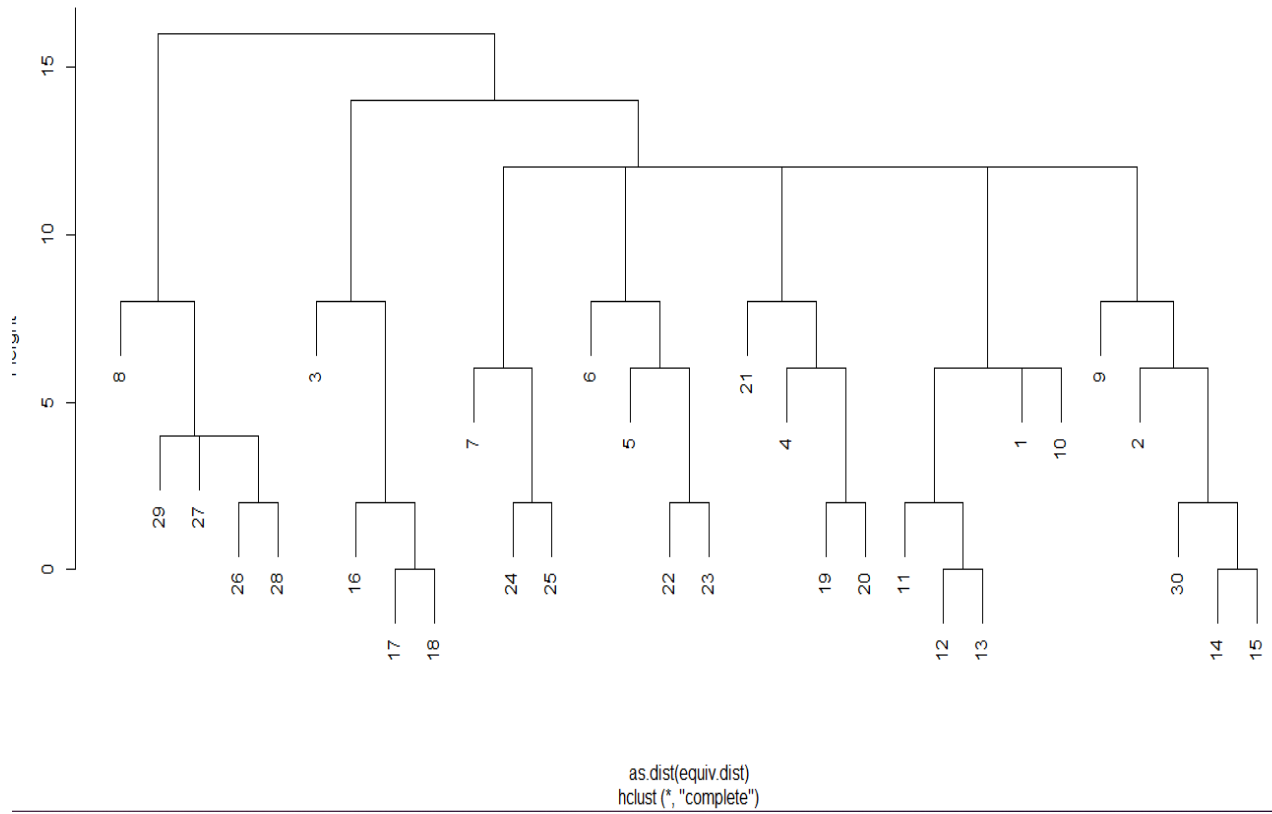
b<-blockmodel(links2,eq,h=10)

plot(b)
```

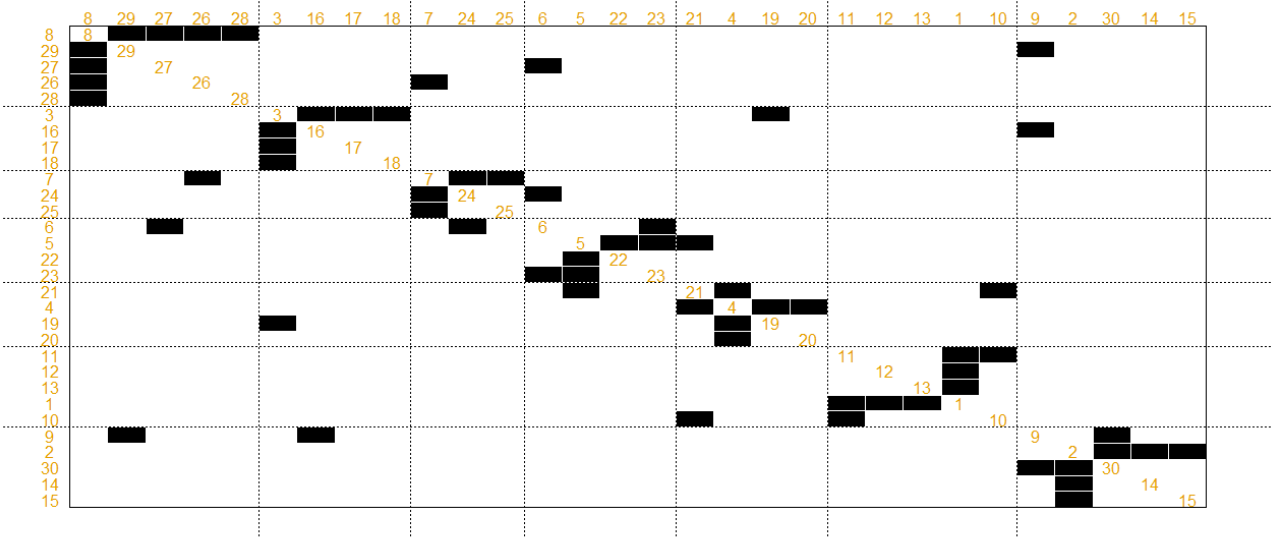
Output:

```
> library(sna)
> library(igraph)
> links2 <- read.csv("C:/Temp/input_files/edges1.csv", header=T, row.names=1)
> eq<-equiv.clust(links2)
> plot(eq)
>
> g.se<-sedist(links2)
>
> plot(cmdscale(as.dist(g.se)))
> b<-blockmodel(links2,eq,h=10)
> plot(b)
>
```

Cluster Dendrogram



Relation - 1



Practical 7

Aim: Perform SVD analysis of a network.

Software(s) used:

- ◆ R ver. 4.1.3
- ◆ RStudio ver. 2022.02.0+433

External packages required:

- ◆ igraph

Description:

- **The igraph package:** igraph is a library and R package for network analysis.
The main goals of the igraph library is to provide a set of data types and functions for:
 - ◆ pain-free implementation of graph algorithms,
 - ◆ fast handling of large graphs, with millions of vertices and edges, allowing rapid prototyping via high level languages like R.
- `matrix()` `matrix()` creates a matrix from the given set of values.
- `c()`: Combines values into a vector or list.
- `print()` `print()` prints its argument and returns it invisibly.

- `svd()`: Compute the singular-value decomposition of a rectangular matrix.

Source Code:

```
library(igraph)
a <- matrix(c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
             1), 9, 4)
print(a)
svd(a)
```

Output:

```
> library(igraph)
> a<- matrix(c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
+             0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1), 9, 4)
> print(a)
      [,1] [,2] [,3] [,4]
[1,]    1    1    0    0
[2,]    1    1    0    0
[3,]    1    1    0    0
[4,]    1    0    1    0
[5,]    1    0    1    0
[6,]    1    0    1    0
[7,]    1    0    0    1
[8,]    1    0    0    1
[9,]    1    0    0    1
```

```

> svd(a)
$d
[1] 3.464102e+00 1.732051e+00 1.732051e+00 1.922963e-16

$u
      [,1]      [,2]      [,3]      [,4]
[1,] -0.3333333  0.4714045 -1.741269e-16  7.760882e-01
[2,] -0.3333333  0.4714045 -3.692621e-16 -1.683504e-01
[3,] -0.3333333  0.4714045 -5.301858e-17 -6.077378e-01
[4,] -0.3333333 -0.2357023 -4.082483e-01  6.774193e-17
[5,] -0.3333333 -0.2357023 -4.082483e-01  6.774193e-17
[6,] -0.3333333 -0.2357023 -4.082483e-01  6.774193e-17
[7,] -0.3333333 -0.2357023  4.082483e-01  5.194768e-17
[8,] -0.3333333 -0.2357023  4.082483e-01  5.194768e-17
[9,] -0.3333333 -0.2357023  4.082483e-01  5.194768e-17

$v
      [,1]      [,2]      [,3] [,4]
[1,] -0.8660254  0.0000000 -4.378026e-17  0.5
[2,] -0.2886751  0.8164966 -2.509507e-16 -0.5
[3,] -0.2886751 -0.4082483 -7.071068e-01 -0.5
[4,] -0.2886751 -0.4082483  7.071068e-01 -0.5

```