NAME: SUNIL.DIBAKAR.PANIGRAHI

ROLL NUMBER: 509

COURSE: MSc CS

SUBJECT: SOCIAL NETWORK ANALYSIS

PRACTICAL: 1-8

INDEX

	DATE	TITLE	SIGN
1		**Write a program to compute the following for a given a network: (i) number of edges, (ii) number of nodes; (iii) degree of node; (iv) node with lowest degree; (v) the adjacency list; (vi) matrix of the graph.**	
2		**Perform following tasks: (i) View data collection forms and/or import onemode/two-mode datasets; (ii) Basic Networks matrices transformations**	
3		**Compute the following node level measures: (i) Density; (ii) Degree; (iii) Reciprocity; (iv) Transitivity; (v) Centralization; (vi) Clustering.**	
4		**For a given network find the following: (i) Length of the shortest path from a given node to another node; (ii) the density of the graph**	
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.**	
6		**Write a program to exhibit structural equivalence, automorphic equivalence, and regular equivalence from a network.**	
7		**Create sociograms for the persons-by-persons network and the committee-bycommittee network for a given relevant problem. Create one-mode**	
8		**Perform SVD analysis of a network.**	

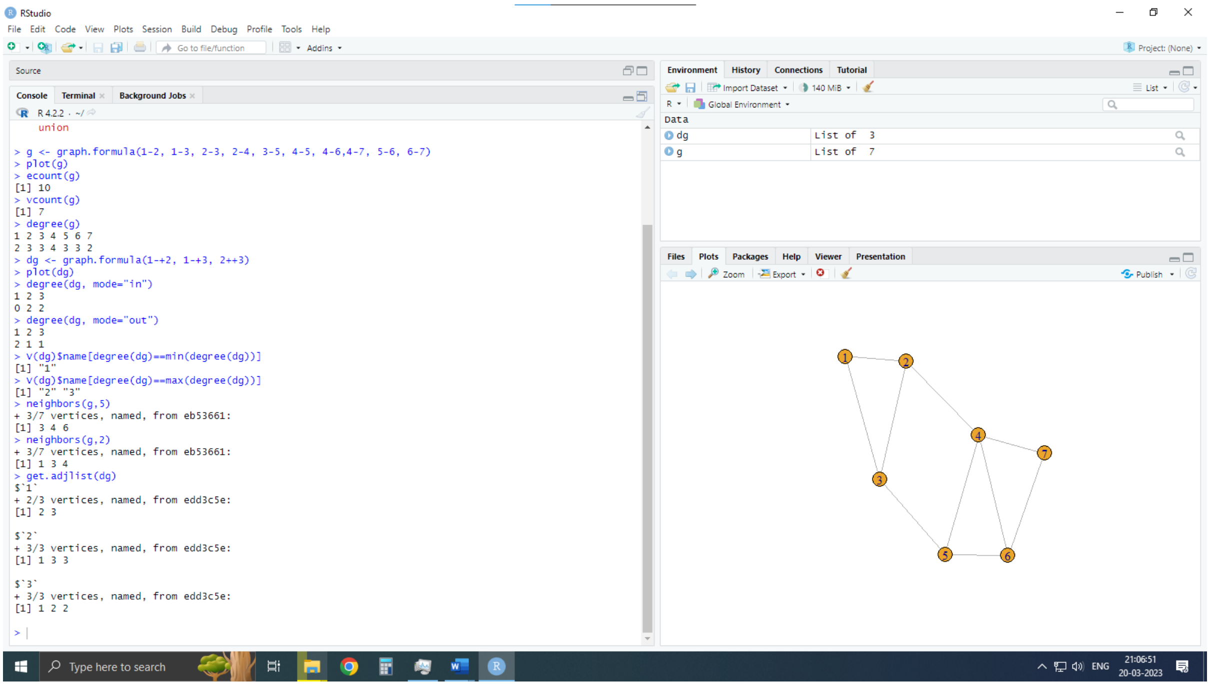
Practical 1: Write a program to compute the following for a given a network: (i) number of edges, (ii) number of nodes; (iii) degree of node; (iv) node with lowest degree; (v) the adjacency list; (vi) matrix of the graph.

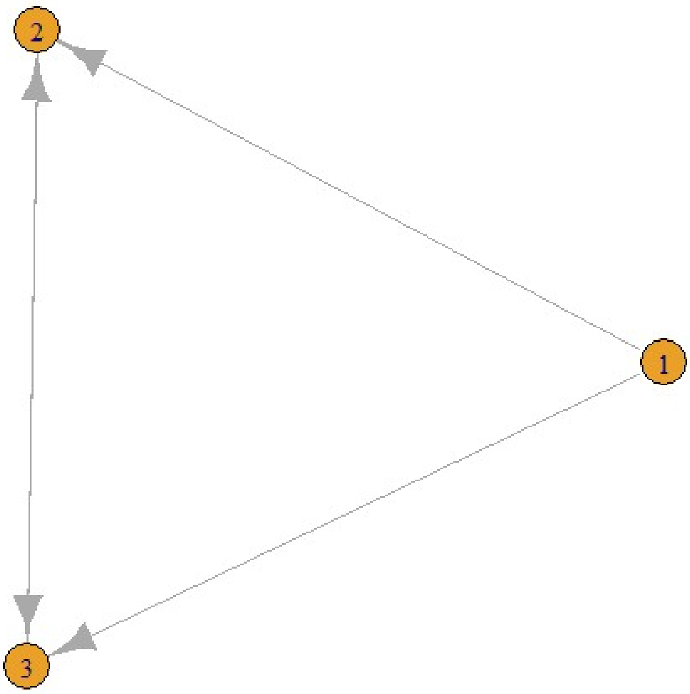
Code:

```
# Install and load the igraph package
install.packages("igraph")
library(igraph)

# Create a graph object 'g' using graph.formula function with edges 1-
2, 1-3, 2-3, 2-4, 3-5, 4-5, 4-6, 4-7, 5-6, 6-7
g <- graph.formula(1-2, 1-3, 2-3, 2-4, 3-5, 4-5, 4-6, 4-7, 5-6, 6-7)

# Plot the graph object 'g'
plot(g)

# Count the number of edges in 'g'
ecount(g)

# Count the number of vertices in 'g'
vcount(g)

# Calculate the degree of each vertex in 'g'
degree(g)

# Create another graph object 'dg' using graph.formula function with
edges 1->2, 1->3, 2<-3
dg <- graph.formula(1-+2, 1-+3, 2++3)

# Plot the graph object 'dg'
plot(dg)

# Calculate the in-degree of each vertex in 'dg'
degree(dg, mode="in")

# Calculate the out-degree of each vertex in 'dg'
degree(dg, mode="out")

# Print the name of the vertex with the minimum degree in 'dg'
V(dg)$name[degree(dg)==min(degree(dg))]

# Print the name of the vertex with the maximum degree in 'dg'
V(dg)$name[degree(dg)==max(degree(dg))]
```

```r
# Find the neighbors of vertex 5 in 'g'
neighbors(g,5)

# Find the neighbors of vertex 2 in 'g'
neighbors(g,2)

# Get the adjacency list of 'dg'
get.adjlist(dg)

# Get the adjacency matrix of 'g'
get.adjacency(g)
```

OUTPUT


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Project: (None)
Source
Console Terminal Background Jobs
R 4.2.2 · ~/
union
> g <- graph.formula(1-2, 1-3, 2-3, 2-4, 3-5, 4-5, 4-6,4-7, 5-6, 6-7)
> plot(g)
> ecount(g)
[1] 10
> vcount(g)
[1] 7
> degree(g)
1 2 3 4 5 6 7
2 3 3 4 3 3 2
> dg <- graph.formula(1-+2, 1-+3, 2++3)
> plot(dg)
> degree(dg, mode="in")
1 2 3
0 2 2
> degree(dg, mode="out")
1 2 3
2 1 1
> V(dg)$name[degree(dg)==min(degree(dg))]
[1] "1"
> V(dg)$name[degree(dg)==max(degree(dg))]
[1] "2" "3"
> neighbors(g,5)
+ 3/7 vertices, named, from eb53661:
[1] 3 4 6
> neighbors(g,2)
+ 3/7 vertices, named, from eb53661:
[1] 1 3 4
> get.adjlist(dg)
$`1`
+ 2/3 vertices, named, from edd3c5e:
[1] 2 3
$`2`
+ 3/3 vertices, named, from edd3c5e:
[1] 1 3 3
$`3`
+ 3/3 vertices, named, from edd3c5e:
[1] 1 2 2
Environment History Connections Tutorial
Import Dataset
140 MiB
List
Global Environment
Data
dg List of 3
g List of 7
Files Plots Packages Help Viewer Presentation
Zoom
Export
Publish
1
2
3
4
5
6
7
Type here to search
ENG
21:06:51
20-03-2023

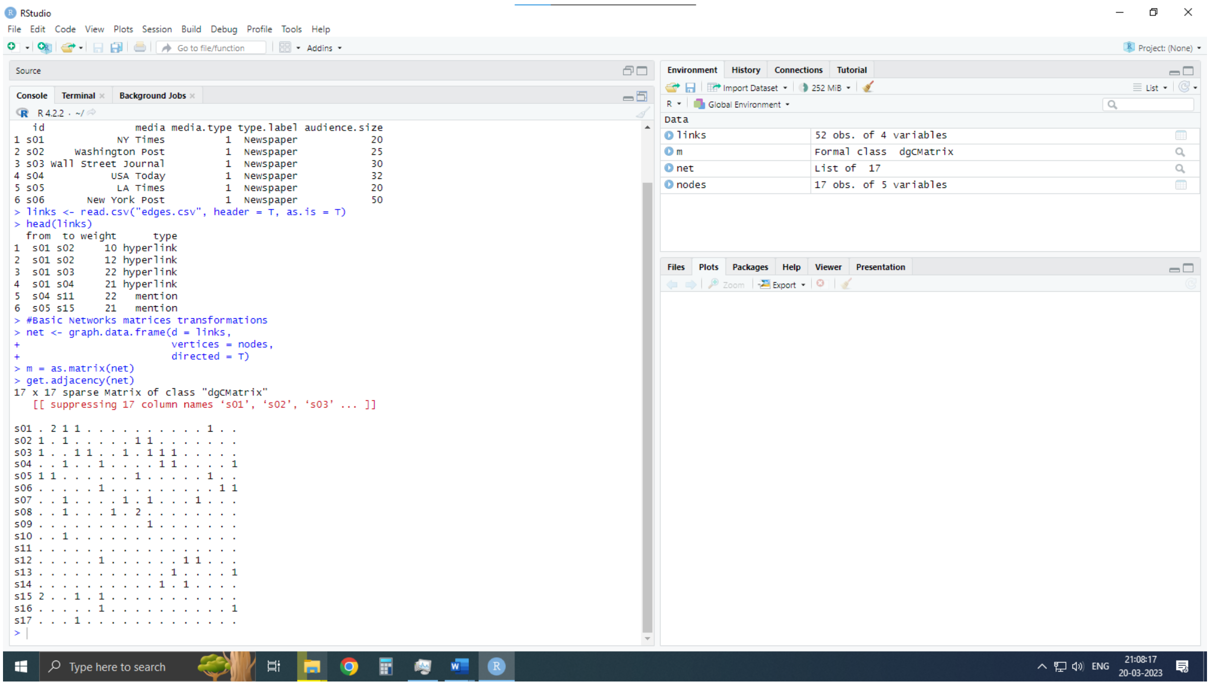

2
1
3

Practical 2: Perform following tasks: (i) View data collection forms and/or import onemode/two-mode datasets; (ii) Basic Networks matrices transformations

Code:

```
# Get the current working directory
getwd()

# Set the working directory to "d:/SNA_pract"
setwd("d:/SNA_pract")

# Read the nodes.csv file into a data frame 'nodes'
nodes <- read.csv("nodes.csv", header = T, , as.is = T)

# Print the first few rows of 'nodes'
head(nodes)

# Read the edges.csv file into a data frame 'links'
links <- read.csv("edges.csv", header = T, as.is = T)

# Print the first few rows of 'links'
head(links)

# Create a graph object 'net' from the data frames 'nodes' and 'links'
net <- graph.data.frame(d = links,
                        vertices = nodes,
                        directed = T)

# Convert the graph object 'net' to an adjacency matrix 'm'
m = as.matrix(net)

# Print the adjacency matrix of 'net'
get.adjacency(net)

# Plot the graph object 'net'
plot(net)
```

OUTPUT


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Source
Console
Terminal
Background Jobs
R 4.2.2 · ~/
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
> links <- read.csv("edges.csv", header = T, as.is = T)
> head(links)
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
> #Basic Networks matrices transformations
> net <- graph.data.frame(d = links,
+ vertices = nodes,
+ directed = T)
> m = as.matrix(net)
> get.adjacency(net)
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 . . . .
Environment History Connections Tutorial
Import Dataset
252 MiB
Global Environment
List
Data
links 52 obs. of 4 variables
m Formal class dgCMatrix
net List of 17
nodes 17 obs. of 5 variables
Files Plots Packages Help Viewer Presentation
Zoom
Export
Project: (None)
Type here to search
ENG
21:08:17
20-03-2023

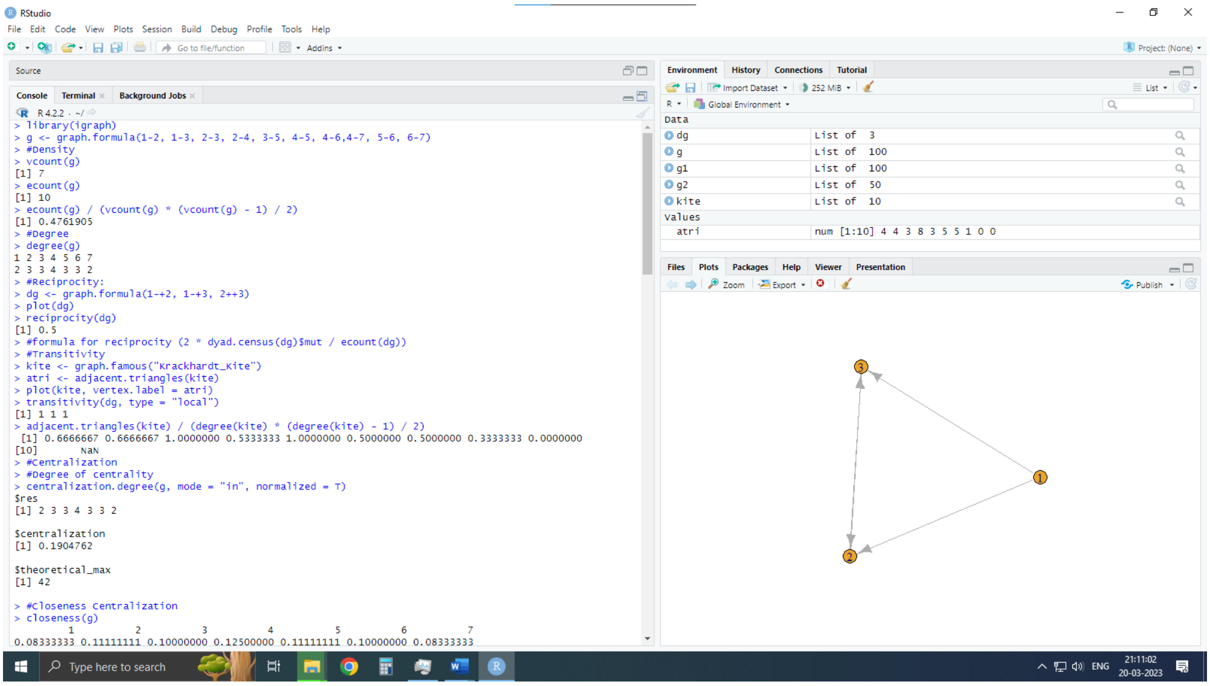
Practical 3: Compute the following node level measures: (i) Density; (ii) Degree; (iii) Reciprocity; (iv) Transitivity; (v) Centralization; (vi) Clustering.

Code:

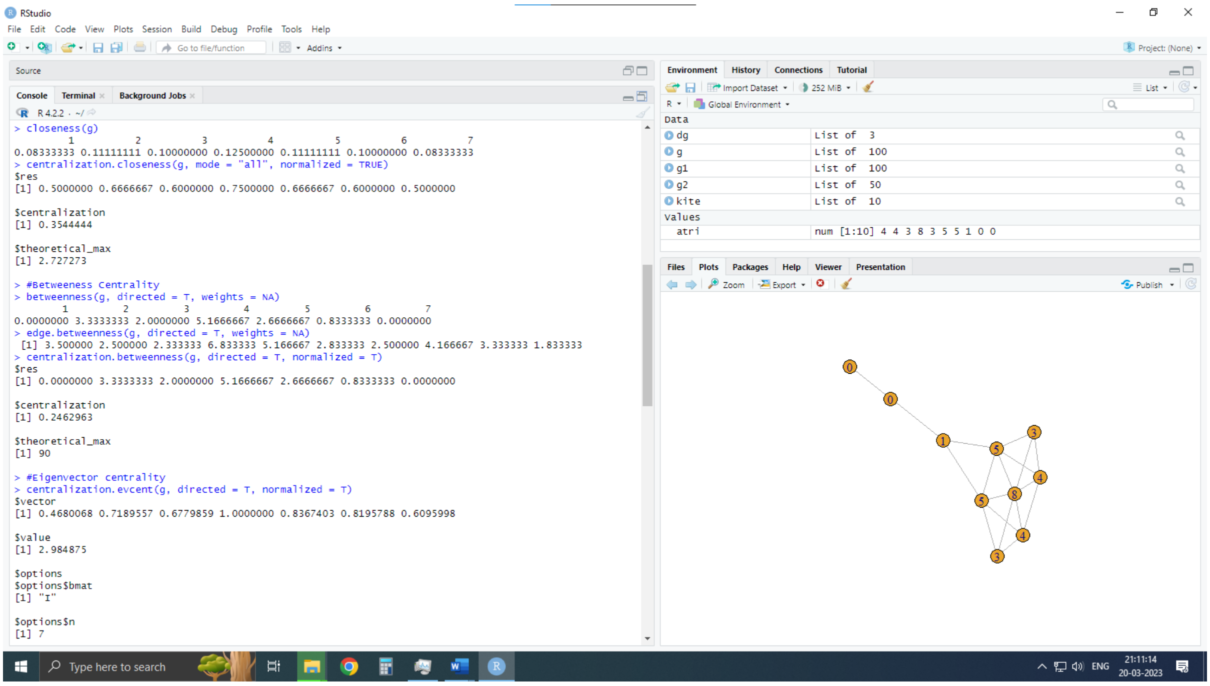
```
# Load the igraph library
library(igraph)

# Create a graph object 'g'
g <- graph.formula(1-2, 1-3, 2-3, 2-4, 3-5, 4-5, 4-6,4-7, 5-6, 6-7)

# Density
# Number of vertices
vcount(g)

# Number of edges
ecount(g)

# Density of the graph
ecount(g) / (vcount(g) * (vcount(g) - 1) / 2)

# Degree
degree(g)

# Reciprocity:
# Create a directed graph 'dg'
dg <- graph.formula(1-+2, 1-+3, 2++3)

# Plot the directed graph 'dg'
plot(dg)

# Reciprocity of the directed graph 'dg'
reciprocity(dg)

# Formula for reciprocity
(2 * dyad.census(dg)$mut / ecount(dg))

# Transitivity
# Create a famous graph 'kite'
kite <- graph.famous("Krackhardt_Kite")
```

```r
# Find the adjacent triangles in the 'kite' graph
atri <- adjacent.triangles(kite)

# Plot the 'kite' graph with vertex labels as adjacent triangles
plot(kite, vertex.label = atri)

# Local transitivity of the directed graph 'dg'
transitivity(dg, type = "local")

# Proportion of adjacent triangles to all possible triangles in the
'kite' graph
adjacent.triangles(kite) / (degree(kite) * (degree(kite) - 1) / 2)

# Centralization
# Degree of centrality
centralization.degree(g, mode = "in", normalized = T)

# Closeness Centralization
closeness(g)
centralization.closeness(g, mode = "all", normalized = TRUE)

# Betweeness Centrality
betweenness(g, directed = T, weights = NA)
edge.betweenness(g, directed = T, weights = NA)
centralization.betweenness(g, directed = T, normalized = T)

# Eigenvector centrality
centralization.evcent(g, directed = T, normalized = T)

# Clustering
# Create two graphs 'g1' and 'g2'
g2 <- barabasi.game(50, p = 2, directed = F)
g1 <- watts.strogatz.game(1, size = 100, nei = 5, p = 0.05)

# Combine the two graphs 'g1' and 'g2'
g <- graph.union(g1, g2)

# Simplify the combined graph 'g'
g <- simplify(g)

# Plot the simplified graph 'g'
plot(g)
```

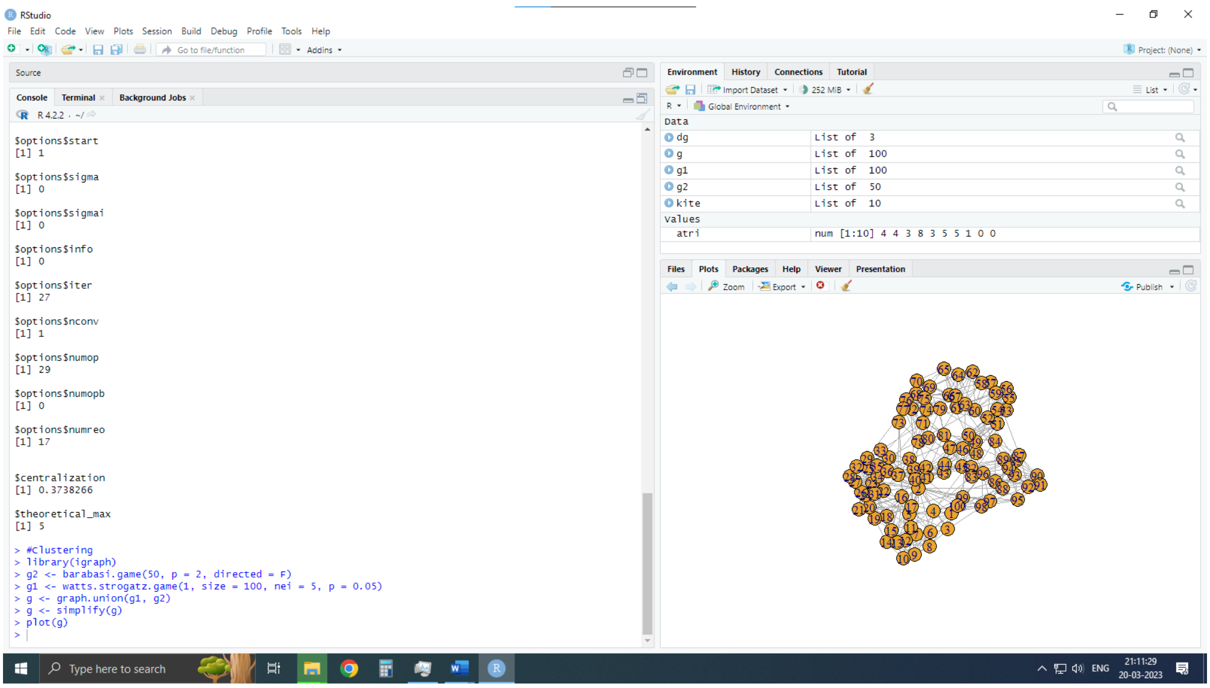
OUTPUT



RStudio
> library(igraph)
> g <- graph.formula(1-2, 1-3, 2-3, 2-4, 3-5, 4-5, 4-6,4-7, 5-6, 6-7)
> #Density
> vcount(g)
[1] 7
> ecount(g)
[1] 10
> ecount(g) / (vcount(g) * (vcount(g) - 1) / 2)
[1] 0.4761905
> #Degree
> degree(g)
1 2 3 4 5 6 7
2 3 3 4 3 3 2
> #Reciprocity:
> dg <- graph.formula(1-+2, 1-+3, 2++3)
> plot(dg)
> reciprocity(dg)
[1] 0.5
> #formula for reciprocity (2 * dyad.census(dg)$mut / ecount(dg))
> #Transitivity
> kite <- graph.famous("Krackhardt_Kite")
> atri <- adjacent.triangles(kite)
> plot(kite, vertex.label = atri)
> transitivity(dg, type = "local")
[1] 1 1 1
> 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
> #Centralization
> #Degree of centrality
> centralization.degree(g, mode = "in", normalized = T)
$res
[1] 2 3 3 4 3 3 2
$centralization
[1] 0.1904762
$theoretical_max
[1] 42
> #Closeness Centralization
> closeness(g)
1 2 3 4 5 6 7
0.08333333 0.11111111 0.10000000 0.12500000 0.11111111 0.10000000 0.08333333
Data
dg List of 3
g List of 100
g1 List of 100
g2 List of 50
kite List of 10
Values
atri num [1:10] 4 4 3 8 3 5 5 1 0 0
3
1
2



RStudio
> closeness(g)
1 2 3 4 5 6 7
0.08333333 0.11111111 0.10000000 0.12500000 0.11111111 0.10000000 0.08333333
> centralization.closeness(g, mode = "all", normalized = TRUE)
$res
[1] 0.5000000 0.6666667 0.6000000 0.7500000 0.6666667 0.6000000 0.5000000
$centralization
[1] 0.3544444
$theoretical_max
[1] 2.727273
> #Betweeness Centrality
> betweenness(g, directed = T, weights = NA)
1 2 3 4 5 6 7
0.0000000 3.3333333 2.0000000 5.1666667 2.6666667 0.8333333 0.0000000
> edge.betweenness(g, directed = T, weights = NA)
[1] 3.500000 2.500000 2.333333 6.833333 5.166667 2.833333 2.500000 4.166667 3.333333 1.833333
> centralization.betweenness(g, directed = T, normalized = T)
$res
[1] 0.0000000 3.3333333 2.0000000 5.1666667 2.6666667 0.8333333 0.0000000
$centralization
[1] 0.2462963
$theoretical_max
[1] 90
> #Eigenvector centrality
> centralization.evcent(g, directed = T, normalized = T)
$vector
[1] 0.4680068 0.7189557 0.6779859 1.0000000 0.8367403 0.8195788 0.6095998
$value
[1] 2.984875
$options
$options$bmat
[1] "I"
$options$n
[1] 7
Data
dg List of 3
g List of 100
g1 List of 100
g2 List of 50
kite List of 10
Values
atri num [1:10] 4 4 3 8 3 5 5 1 0 0


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Project: (None)
Source
Console
Terminal
Background Jobs
R 4.2.2 · ~/
$options$start
[1] 1
$options$sigma
[1] 0
$options$sigmai
[1] 0
$options$info
[1] 0
$options$iter
[1] 27
$options$nconv
[1] 1
$options$numop
[1] 29
$options$numopb
[1] 0
$options$numreo
[1] 17
$centralization
[1] 0.3738266
$theoretical_max
[1] 5
> #Clustering
> library(igraph)
> g2 <- barabasi.game(50, p = 2, directed = F)
> g1 <- watts.strogatz.game(1, size = 100, nei = 5, p = 0.05)
> g <- graph.union(g1, g2)
> g <- simplify(g)
> plot(g)
>
Environment
History
Connections
Tutorial
Import Dataset
252 MiB
List
Global Environment
Data
dg List of 3
g List of 100
g1 List of 100
g2 List of 50
kite List of 10
Values
atri num [1:10] 4 4 3 8 3 5 5 1 0 0
Files
Plots
Packages
Help
Viewer
Presentation
Zoom
Export
Publish
Type here to search
ENG
21:11:29
20-03-2023

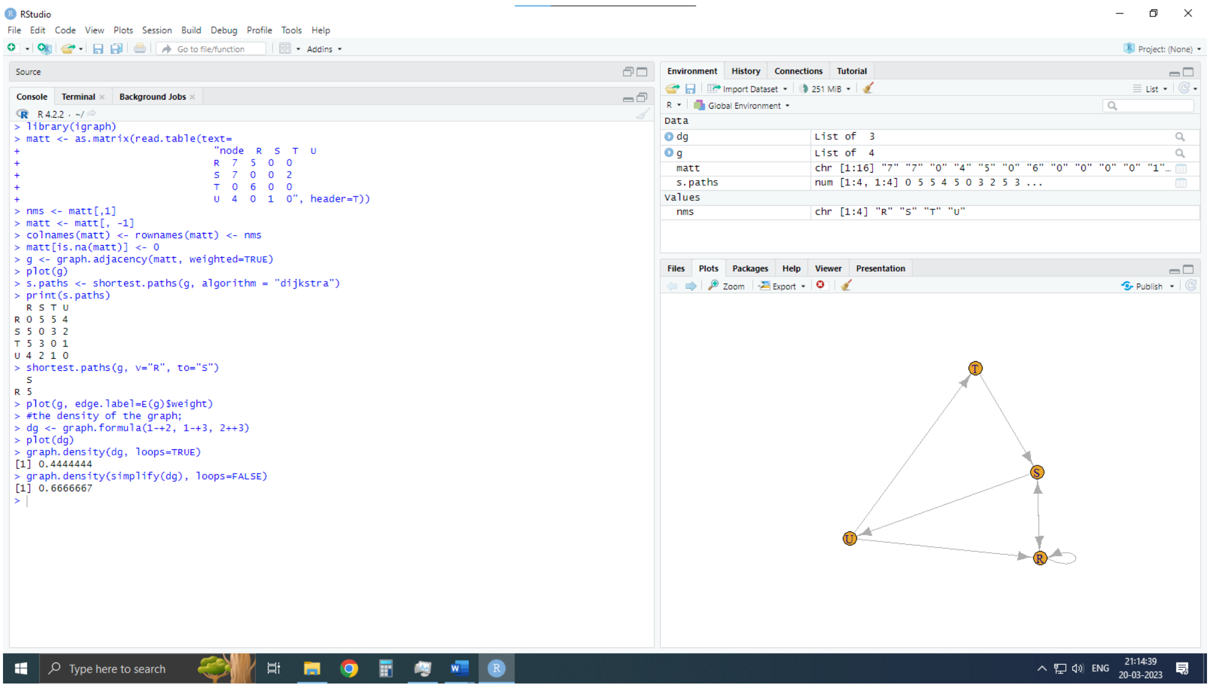
Practical 4: For a given network find the following: (i) Length of the shortest path from a given node to another node; (ii) the density of the graph

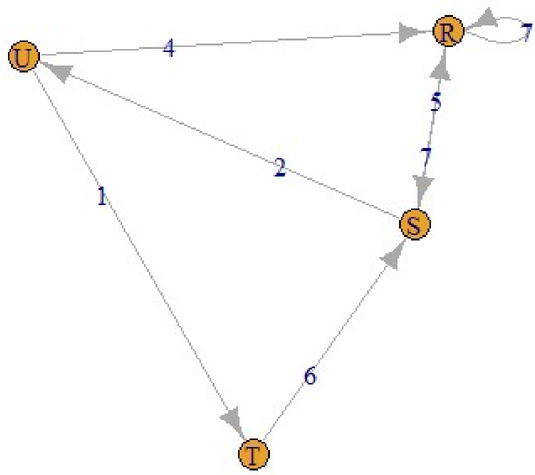
Code:

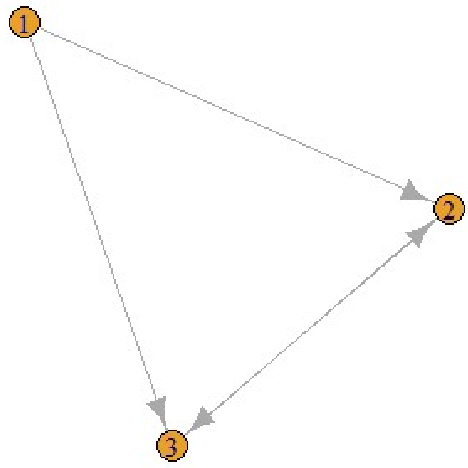
```
library(igraph)

# creating a matrix from a table
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))

# storing the row names in nms and removing the first column
nms <- matt[,1]
matt <- matt[, -1]

# setting the column and row names to be the same
colnames(matt) <- rownames(matt) <- nms

# replacing NA values with 0
matt[is.na(matt)] <- 0

# creating a weighted graph from the matrix
g <- graph.adjacency(matt, weighted=TRUE)

# plotting the graph
plot(g)

# calculating the shortest paths between all pairs of nodes
s.paths <- shortest.paths(g, algorithm = "dijkstra")
print(s.paths)

# calculating the shortest path between R and S
shortest.paths(g, v="R", to="S")

# plotting the graph with edge weights as labels
plot(g, edge.label=E(g)$weight)

# calculating the density of the graph
dg <- graph.formula(1-+2, 1-+3, 2++3)
plot(dg)
graph.density(dg, loops=TRUE)
```

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

OUTPUT





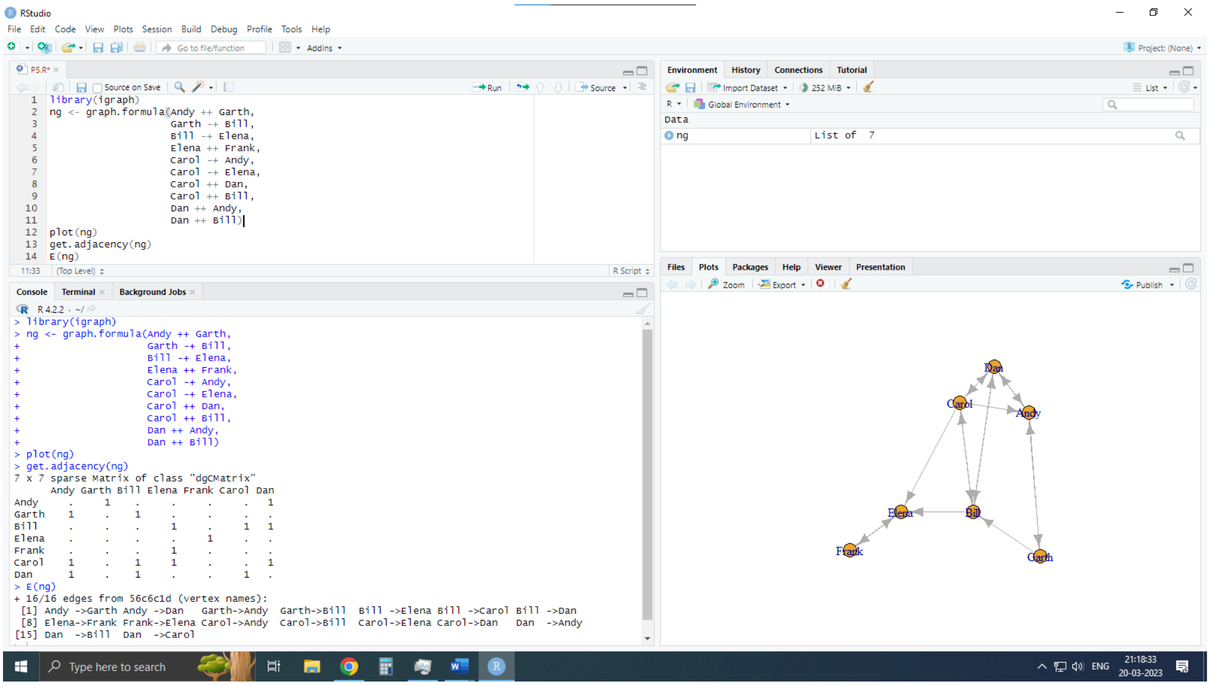

1
2
3

Practical 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.

Code:

```
# Load igraph package
library(igraph)

# Define network using graph formula notation
ng <- graph.formula(Andy ++ Garth,
                    Garth -+ Bill,
                    Bill -+ Elena,
                    Elena ++ Frank,
                    Carol -+ Andy,
                    Carol -+ Elena,
                    Carol ++ Dan,
                    Carol ++ Bill,
                    Dan ++ Andy,
                    Dan ++ Bill)

# Plot the network
plot(ng)

# Display adjacency matrix of the network
get.adjacency(ng)

# Display edges of the network
E(ng)
```

OUTPUT

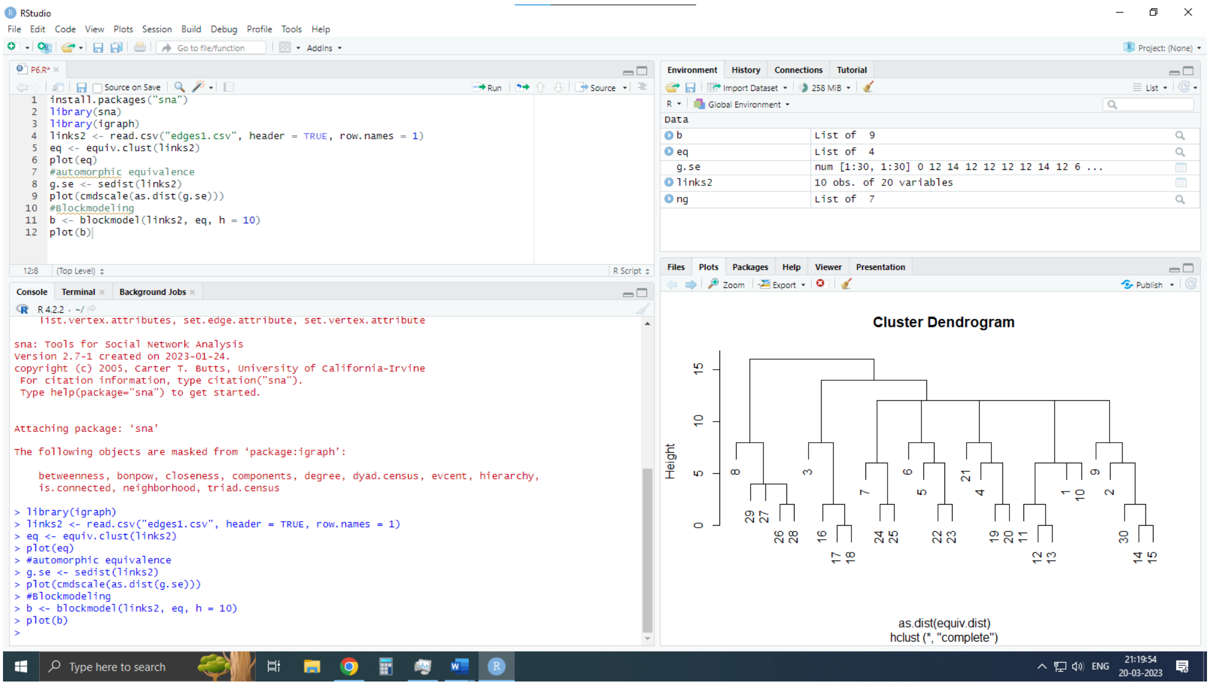

RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
P5.R*
library(igraph)
ng <- graph.formula(Andy ++ Garth,
Garth -+ Bill,
Bill -+ Elena,
Elena ++ Frank,
Carol -+ Andy,
Carol -+ Elena,
Carol ++ Dan,
Carol ++ Bill,
Dan ++ Andy,
Dan ++ Bill)
plot(ng)
get.adjacency(ng)
E(ng)
Console Terminal Background Jobs
R 4.2.2 · ~/
> library(igraph)
> ng <- graph.formula(Andy ++ Garth,
+ Garth -+ Bill,
+ Bill -+ Elena,
+ Elena ++ Frank,
+ Carol -+ Andy,
+ Carol -+ Elena,
+ Carol ++ Dan,
+ Carol ++ Bill,
+ Dan ++ Andy,
+ Dan ++ Bill)
> plot(ng)
> get.adjacency(ng)
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(ng)
+ 16/16 edges from 56c6c1d (vertex names):
[1] Andy ->Garth Andy ->Dan Garth->Andy Garth->Bill Bill ->Elena Bill ->Carol Bill ->Dan
[8] Elena->Frank Frank->Elena Carol->Andy Carol->Bill Carol->Elena Carol->Dan Dan ->Andy
[15] Dan ->Bill Dan ->Carol
Environment History Connections Tutorial
Import Dataset
Global Environment
Data
ng List of 7
Files Plots Packages Help Viewer Presentation
Zoom Export Publish
Dan
Carol
Andy
Elena
Bill
Frank
Garth
Type here to search
ENG
21:18:33
20-03-2023

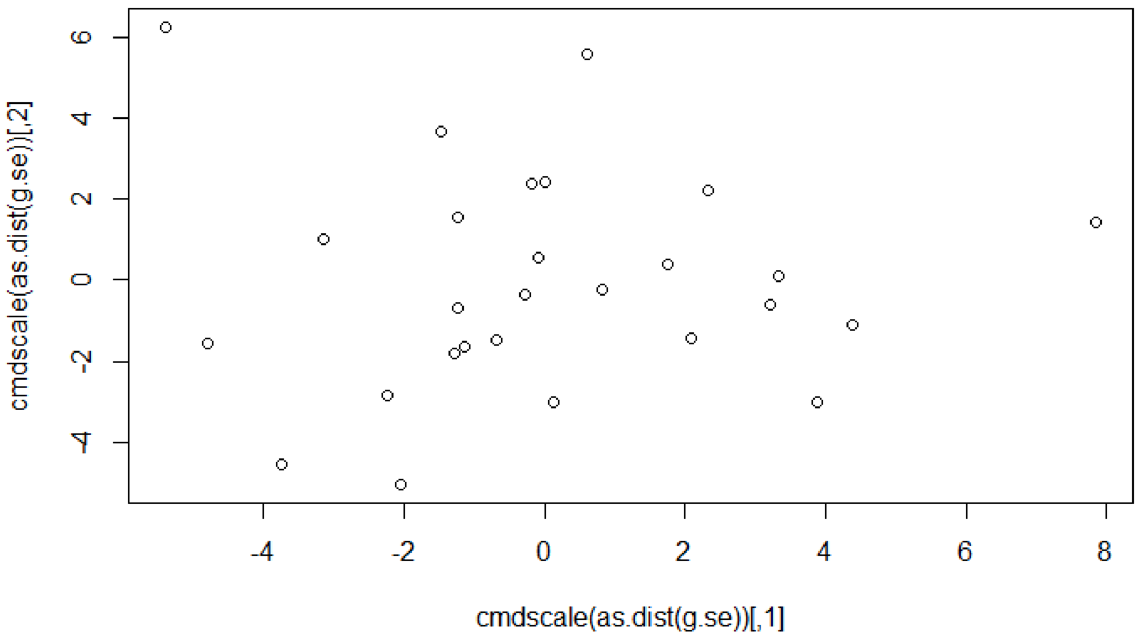
Practical 6: Write a program to exhibit structural equivalence, automorphic equivalence, and regular equivalence from a network.

Codes:

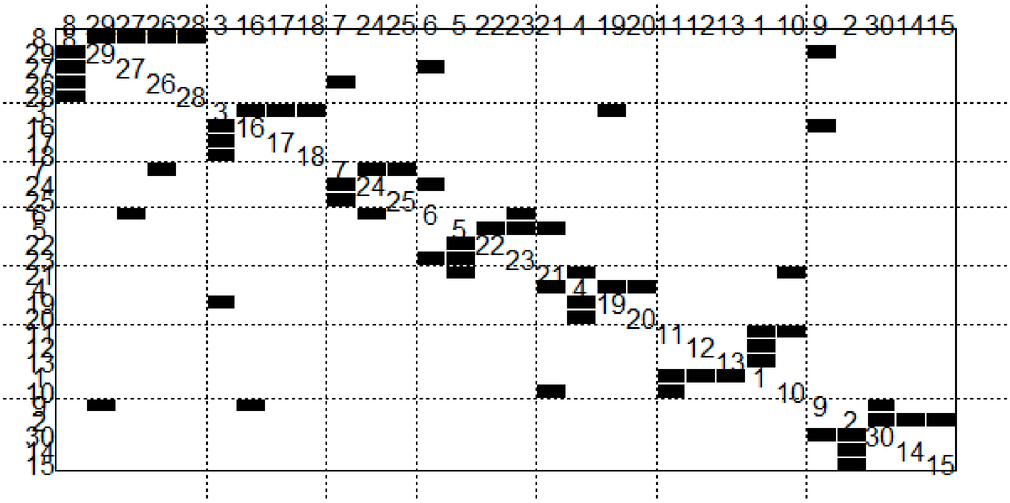
```
# Install and load necessary packages
install.packages("sna")
library(sna)
library(igraph)

# Read data from file
links2 <- read.csv("edges1.csv", header = TRUE, row.names = 1)

# Equivalence clustering
eq <- equiv.clust(links2)
plot(eq)

# Automorphic equivalence
g.se <- sedist(links2)
plot(cmdscale(as.dist(g.se)))

# Blockmodeling
b <- blockmodel(links2, eq, h = 10)
plot(b)
```

OUTPUT


Cluster Dendrogram
Height
as.dist(equiv.dist)
hclust (*, "complete")


cmdscale(as.dist(g.se))[,2]
cmdscale(as.dist(g.se))[,1]

Relation - 1


8 29 27 26 28 3 16 17 18 7 24 25 6 5 22 23 21 4 19 20 11 12 13 1 10 9 2 30 14 15

Practical 7: Perform SVD analysis on network

Code:

```
# Load the igraph library
library(igraph)

# Create a 9x4 matrix with specific values
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 the matrix to the console
print(a)

# Perform singular value decomposition on the matrix
svd(a)
```

OUTPUT

```
> 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
[4] 1.922963e-16

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

$v
           [,1]      [,2]          [,3] [,4]
[1,] -0.8660254 0.0000000 -4.378026e-17  0.5
```