INDEX

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr No.** | **Date** | **Practical Name** | **Pg.**  **No.** | **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. | 2-4 |  |
| 2. |  | Perform following tasks:   1. View data collection forms and/or import one-mode/two- mode datasets; 2. Basic Networks matrices transformations | 5-7 |  |
| 3. |  | Compute the following node level measures: (i) Density; (ii) Degree;(iii) Reciprocity; (iv) Transitivity; (v) Centralization; (vi) Clustering. | 8-16 |  |
| 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; (iii) Draw egocentric network of node G with chosen configuration parameters. | 17-18 |  |
| 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. | 19-21 |  |
| 6. |  | Write a program to exhibit structural equivalence, automatic equivalence, and regular equivalence from a network. | 21-23 |  |
| 7. |  | Create sociograms for the persons-by-persons network and the committee-by- committee network for a given relevant problem. Create one-mode network and two-node network for the same. | 24-26 |  |
| 8. |  | Perform SVD analysis of a network. | 27-28 |  |
| 9. |  | Identify ties within the network using two-mode core periphery analysis. | 29 |  |
| 10. |  | Find “factions” in the network using two-mode faction analysis. | 30-31 |  |

# Practical No 1

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

# number of edges,

1. **number of nodes;**

# degree of node;

1. **node with lowest degree; (v)the adjacency list;**

# (vi) matrix of the graph

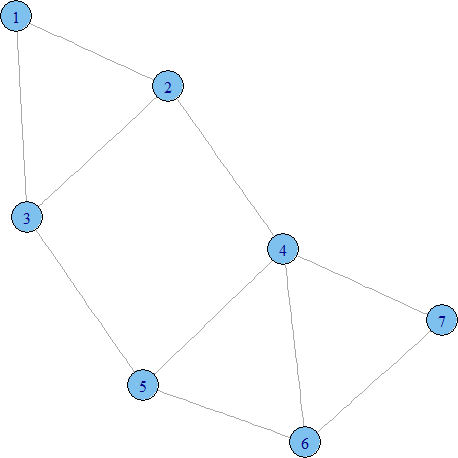
# Step 1: install package

# install.packages(“igraph”)

>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)

>plot(g)



# Number of edges

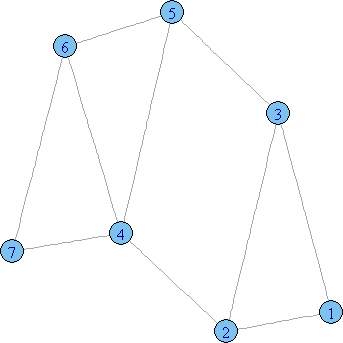
>ecount(g) [1] 10

# No of nodes

>vcount(g)

[1] 7

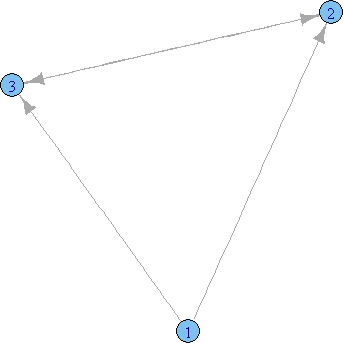
# Degree of nodes



>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

# Node with lowest degree

>V(dg)$name[degree(dg)==min(degree(dg))] [1] "1"

# Node with highest degree

>V(dg)$name[degree(dg)==max(degree(dg))] get[1] "2" "3"

# To find neighbors / adjacency list:

>neighbors(g,5) [1] 3 4 6

>neighbors(g,2) [1] 1 3 4

>get.adjlist(dg)

$`1` [1] 2 3

$`2`

[1] 1 3 3

$`3`

[1] 1 2 2

# Adjacency Matrix

>get.adjacency(g)

7 x 7 sparse Matrix of class "dgCMatrix" 1 2 3 4 5 6 7

1 . 1 1 . . . .

2 1 . 1 1 . . .

3 1 1 . . 1 . .

4 .1 . . 1 1 1

5 . . 1 1 .1 .

6 . . . 1 1 . 1

7 . . . 1 .1 .

# Practical No 2

**Perform following tasks:**

# View data collection forms and/or import one-mode/ two-mode datasets;

1. **Basic Networks matrices transformations.**
2. **View data collection forms and/or import one-mode/ two-mode datasets.**

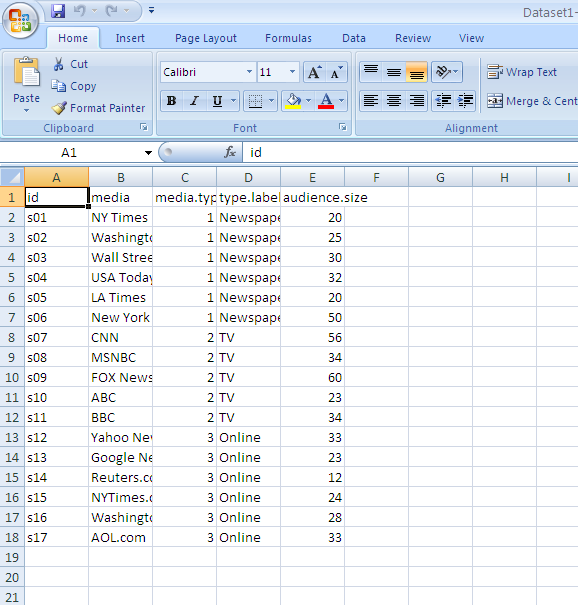
getwd()

[1] "C:/Users/admin/Documents"

>setwd("copy the path from my files don’t write the file name till downloads and save file as csv demilated")

# Reading data from a csv file

>nodes<-read.csv(“cred.csv”, header=T, , as.is=T)



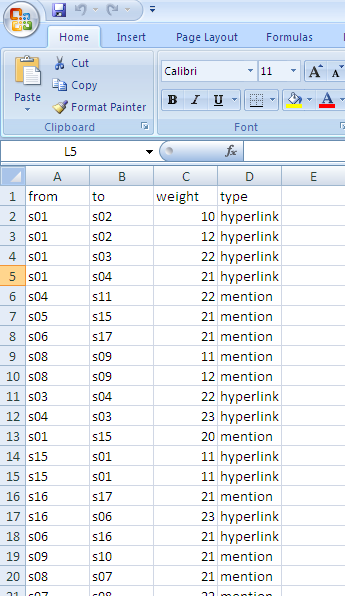
> head(nodes)

Output:-

id media media.typetype.labelaudience.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(“Dataset1-Media-Example-EDGES.csv”, header=T, as.is=T)



>head(links) Output:-

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
7. Basic Networks matrices transformations

install.packages("igraph") library(igraph)

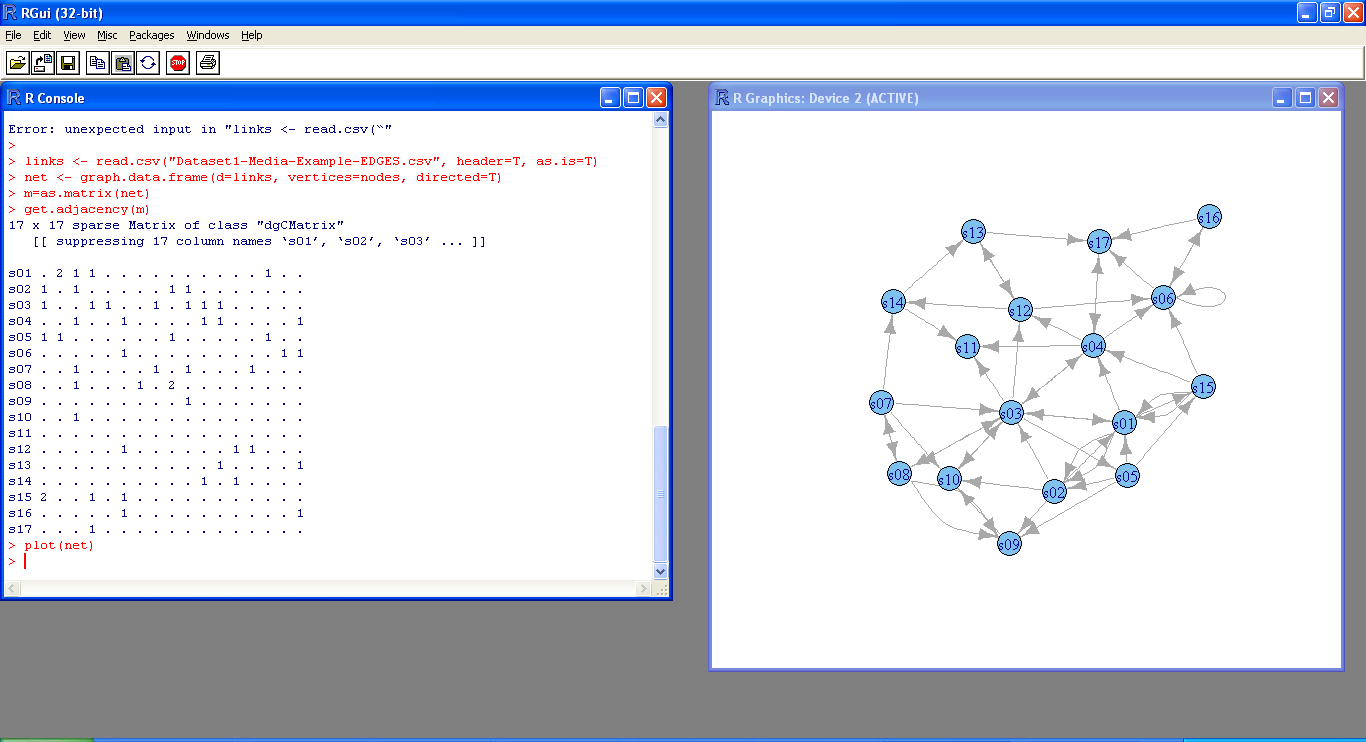
>links <- data.frame(from = c("A", "B", "C"), to = c("B", "C", "A"))

> nodes <- data.frame(name = c("A", "B", "C"))

> links<- read.csv(“cred.csv”, header=T, as.is=T)

>net<- graph\_from\_data\_frame(d=links, directed=TRUE) >get.adjacency(net)

>plot(net)



# Practical No 3 Compute the following node level measures:

1. **Density;**

# Degree;

1. **Reciprocity;**

# Transitivity;

1. **Centralization;**

# Clustering.

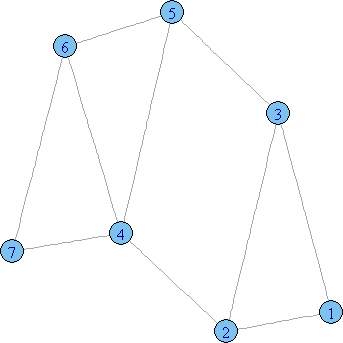
1. **Density**

>vcount(g)

[1] 7

>ecount(g) [1] 10

>ecount(g)/(vcount(g)\*(vcount(g)-1)/2) [1] 0.4719



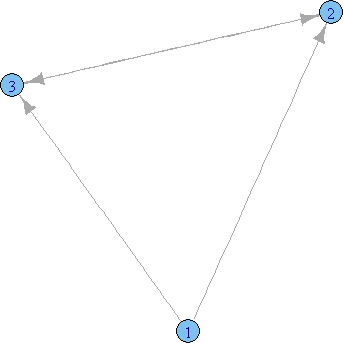
# Degree

>degree(net)

s01 s02 s03 s04 s05 s06 s07 s08 s09 s10 s11 s12 10 7 13 9 5 8 5 6 5 5 3 6

s13 s14 s15 s16 s17 4 4 6 3 5

# Reciprocity:



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

>plot(dg)

>reciprocity(dg) [1] 0.5

# Formula

>dyad.census(dg)

$mut

[1] 1

$asym

[1] 2

$null

[1] 0

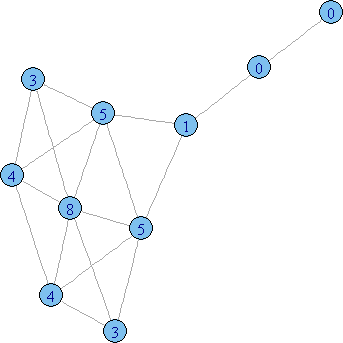
* 2\*dyad.census(dg)$mut/ecount(dg) [1] 0.5

# Transitivity

>kite<- graph.famous("Krackhardt\_Kite")

>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

[7] 0.5000000 0.3333333 0.0000000 NaN

# Formula

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

[1] 0.6666667 0.6666667 1.0000000 0.5333333 1.0000000 0.5000000

[7] 0.5000000 0.3333333 0.0000000 NaN

# Centralization

* + **Degree of centrality**

>centralization.degree(net, mode="in", normalized=T)

$res

[1] 5 3 6 4 1 5 1 2 4 4 3 3 2 2 2 1 4

$centralization [1] 0.1838235

$theoretical\_max [1] 272

# Closeness Centralization

>closeness(net, mode="all", weights=NA)

s01 s02 s03 s04 s05 s06 s07

0.03333333 0.03030303 0.04166667 0.03846154 0.03225806 0.03125000 0.03030303

s08 s09 s10 s11 s12 s13 s14

0.02857143 0.02564103 0.02941176 0.03225806 0.03571429 0.02702703 0.02941176

s15 s16 s17 0.03030303 0.02222222 0.02857143

>centralization.closeness(net, mode="all", normalized=T)

$res

[1] 0.5333333 0.4848485 0.6666667 0.6153846 0.5161290 0.5000000 0.4848485

[8] 0.4571429 0.4102564 0.4705882 0.5161290 0.5714286 0.4324324 0.4705882

[15] 0.4848485 0.3555556 0.4571429

$centralization [1] 0.3753596

$theoretical\_max [1] 7.741935

# Betweeness Centrality

>betweenness(net, directed=T, weights=NA)

|  |  |  |
| --- | --- | --- |
| s01 | s02 s03 s04 | s05 s06 s07 |
| 26.857143 | 6.238095 126.511905 | 92.642857 13.000000 20.333333 1.750000 |
| s08 | s09 s10 s11 | s12 s13 s14 |
| 21.000000 | 1.000000 15.000000 | 0.000000 33.500000 20.000000 4.000000 |
| s15 | s16 s17 |  |
| 5.666667 | 0.000000 58.500000 |  |

>edge.betweenness(net, directed=T, weights=NA)

[1] 6.619048 6.619048 11.785714 8.333333 6.500000 11.166667 21.333333

[8] 4.250000 4.250000 16.000000 64.476190 9.500000 3.261905 3.261905

[15] 15.000000 1.000000 15.000000 17.000000 16.750000 2.000000 1.250000

[22] 8.000000 12.500000 4.000000 26.000000 18.000000 14.500000 17.000000

|  |  |
| --- | --- |
| [29] 7.500000 4.500000 2.738095 23.000000 11.000000 31.000000 | 9.011905 |
| [36] 18.000000 28.500000 0.000000 3.000000 6.500000 17.000000 | 8.666667 |
| [43] 74.500000 11.750000 34.000000 4.500000 6.333333 8.809524 | 5.333333 |
| [50] 3.000000 28.000000 10.000000 |  |

>centralization.betweenness(net, directed=T, normalized=T)

$res

[1] 26.857143 6.238095 126.511905 92.642857 13.000000 20.333333

[7] 1.750000 21.000000 1.000000 15.000000 0.000000 33.500000

[13] 20.000000 4.000000 5.666667 0.000000 58.500000

$centralization [1] 0.4439329

$theoretical\_max [1] 3840

# Eigenvector centrality

>centralization.evcent(net, directed=T, normalized=T)

$vector

[1] 0.7694528 0.5623895 1.0000000 0.8569443 0.3049992 0.9285033 0.1025656

[8] 0.3362816 0.4696841 0.6510633 0.6361813 0.6479337 0.2674341 0.2289017

[15] 0.3277070 0.2831928 0.7125008

$value

[1] 3.278697

$options

$options$bmat

[1] "I"

$options$n [1] 17

$options$which

[1] "LR"

$options$nev

[1] 1

$options$tol

$options$ncv

[1] 0

$options$ldv

[1] 0

$options$ishift

[1] 1

$options$maxiter [1] 3000

$options$nb

[1] 1

$options$mode

[1] 1

$options$start

[1] 1

$options$sigma

[1] 0

$options$sigmai

[1] 0

$options$info

[1] 0

$options$iter

[1] 7

$options$nconv

[1] 1

$options$numop [1] 31

$options$numopb

$options$numreo [1] 18

$centralization [1] 0.4946416

$theoretical\_max [1] 16

# Clustering

>library(igraph)

# let's generate two networks and merge them into one graph.

>g2 <- barabasi.game(50, p=2, directed=F)

>g1 <- watts.strogatz.game(1, size=100, nei=5, p=0.05)

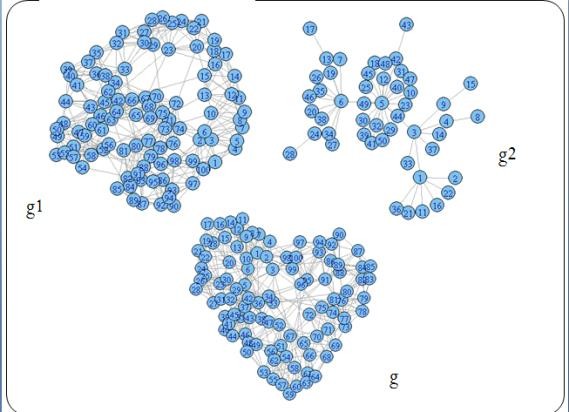
>g <- graph.union(g1,g2)

#Let's remove multi-edges and loops

>g <- simplify(g)

# 1st we calculate the edge betweenness,

>ebc<- edge.betweenness.community(g, directed=F)



>mods<- sapply(0:ecount(g), function(i)

{

g2 <- delete.edges(g, ebc$removed.edges[seq(length=i)]) cl<- clusters(g2)$membership

modularity(g,cl)

})

# Now, let's color the nodes according to their membership

>g2<-delete.edges(g, ebc$removed.edges[seq(length=which.max(mods)-1)])

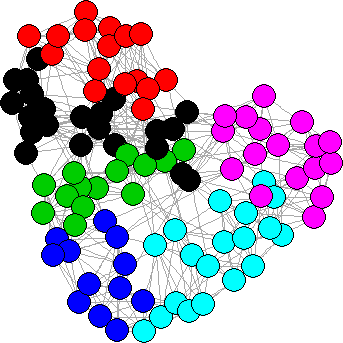
>V(g)$color=clusters(g2)$membership

# Let's choose a layout for the graph

>g$layout<- layout.fruchterman.reingold

# plot it

>plot(g, vertex.label=NA)



# fastgreedy.communityagorithm

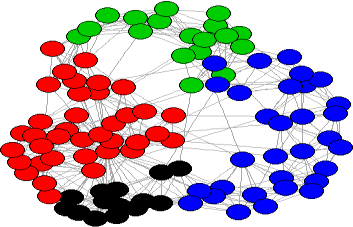
>fc<- fastgreedy.community(g)

>com<-community.to.membership(g, fc$merges, steps= which.max(fc$modularity)-1)

>V(g)$color <- com$membership+1

>g$layout<- layout.fruchterman.reingold

>plot(g, vertex.label=NA)



FASTGREEDY ALGORITHM

# PRACTICAL NO 4

**For a given network find the following:**

# Length of the shortest path from a given node to another node;

1. **The density of the graph;**

# Draw egocentric network of node G with chosen configuration parameters.

1. **Length of the shortest path from a given node to another node;**

>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))

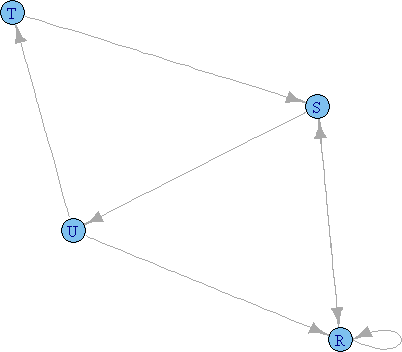
>nms<- matt[,1 ]

>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")

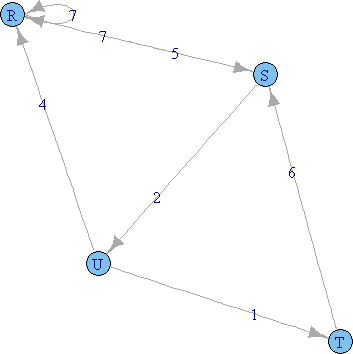
>print(s.paths)

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

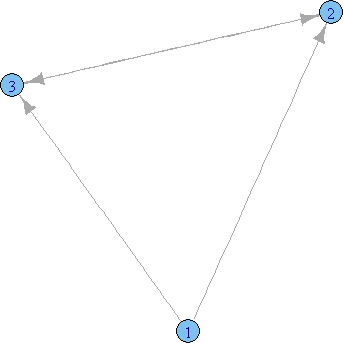
>plot(g, edge.label=E(g)$weight)

# The density of the graph;

>library(igraph)

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

>plot(dg)



>graph.density(dg, loops=TRUE) [1] 0.4444444

* + Without considering loops

>graph.density(simplify(dg), loops=FALSE) [1] 0.6666667

# Practical No 5

**Write a program to distinguish between:**

# a network as a sociogram (or “network graph”)

1. **a network as a matrix,&**

# a network as an edge list.

**Using 3 distinct networks representatives of each.**

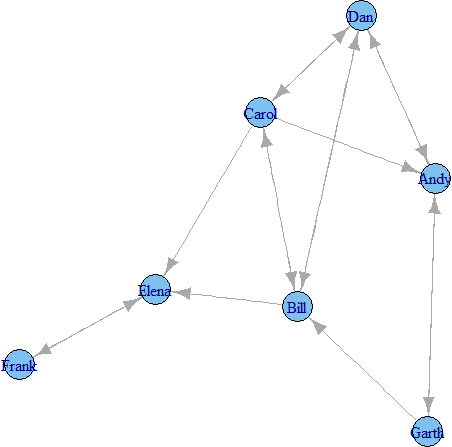
# a network as a sociogram (or “network graph”)

>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)



# a network as a matrix,

>get.adjacency(ng)

1. x 7 sparse Matrix of class "dgCMatrix“ Andy Garth Bill Elena Frank Carol Dan

. . . . 1

|  |  |  |
| --- | --- | --- |
| Andy | . | 1 |
| Garth | 1 | . |

1 . . . .

Bill . . . 1 . 1 1

Elena . . . . 1 . .

Frank . . . 1 . . . Carol 1 .1 1 . . 1

Dan 1 . 1 . . 1 .

# a network as an edge list.

>E(ng)

Edge sequence:

1. Andy -> Garth
2. Andy -> Dan
3. Garth -> Andy
4. Garth -> Bill
5. Bill -> Elena
6. Bill -> Carol
7. Bill -> Dan
8. Elena -> Frank
9. Frank -> Elena
10. Carol -> Andy
11. Carol -> Bill
12. Carol -> Elena
13. Carol -> Dan
14. Dan -> Andy
15. Dan -> Bill
16. Dan -> Carol

>get.adjedgelist(ng,mode="in")

$Andy

[1] 3 10 14

$Garth

[1] 1

$Bill

[1] 4 11 15

$Elena

[1] 5 9 12

$Frank

[1] 8

$Carol [1] 6 16

$Dan

[1] 2 7 13

# Practical No 6

**Write a program to exhibit**

# structural equivalence,

1. **automatic equivalence,**

# regular equivalence from a network.

* 1. **structural equivalence**

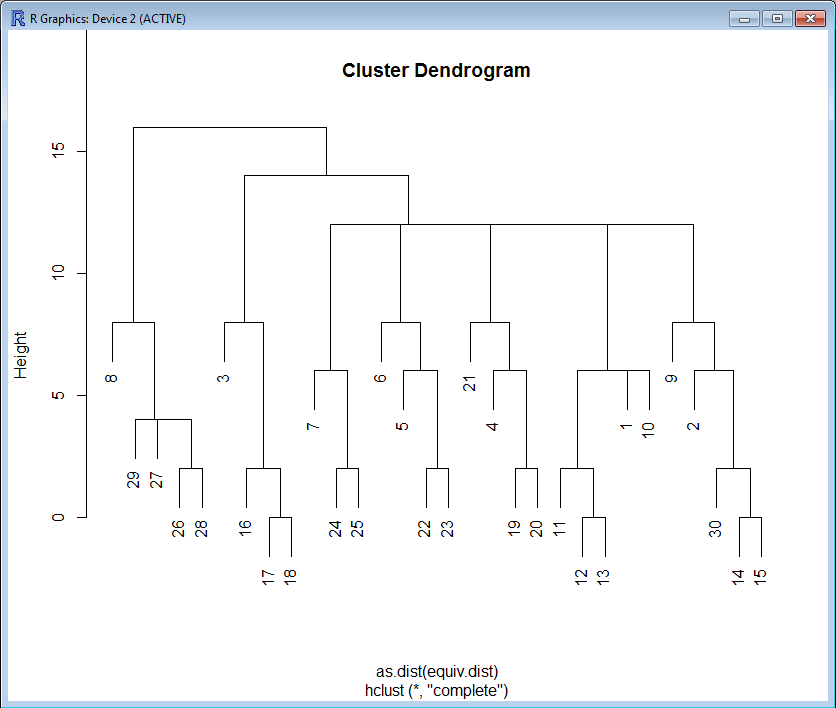
>library(sna)

>library(igraph)

* links2 <- read.csv(“Dataset2-Media-User-Example-EDGES.csv”, header=T, row.names=1)

>eq<-equiv.clust(links2)

>plot(eq)

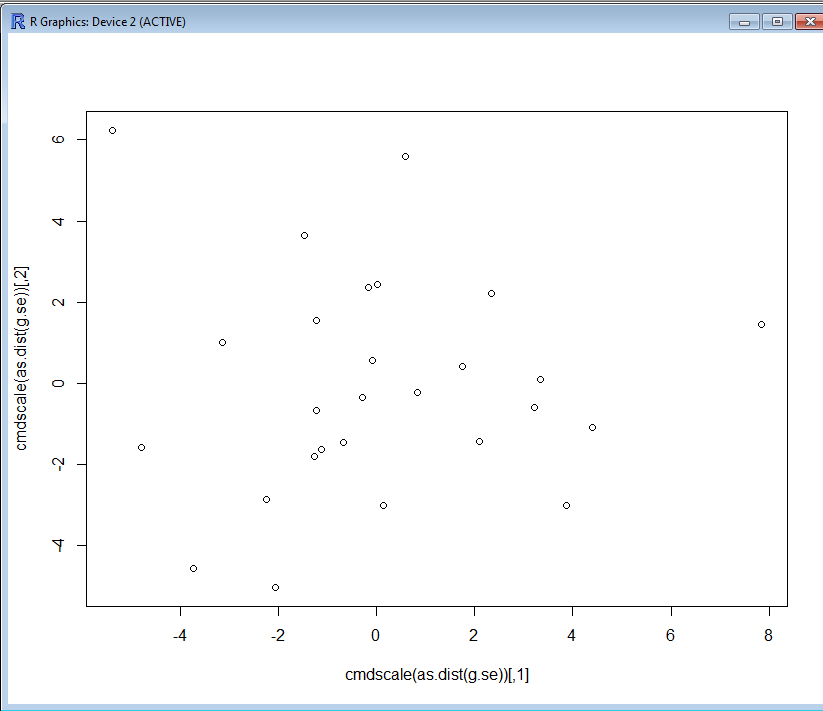


# automatic equivalence,

>g.se<-sedist(links2)

Plot a metric MDS of vertex positions in two dimensions

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

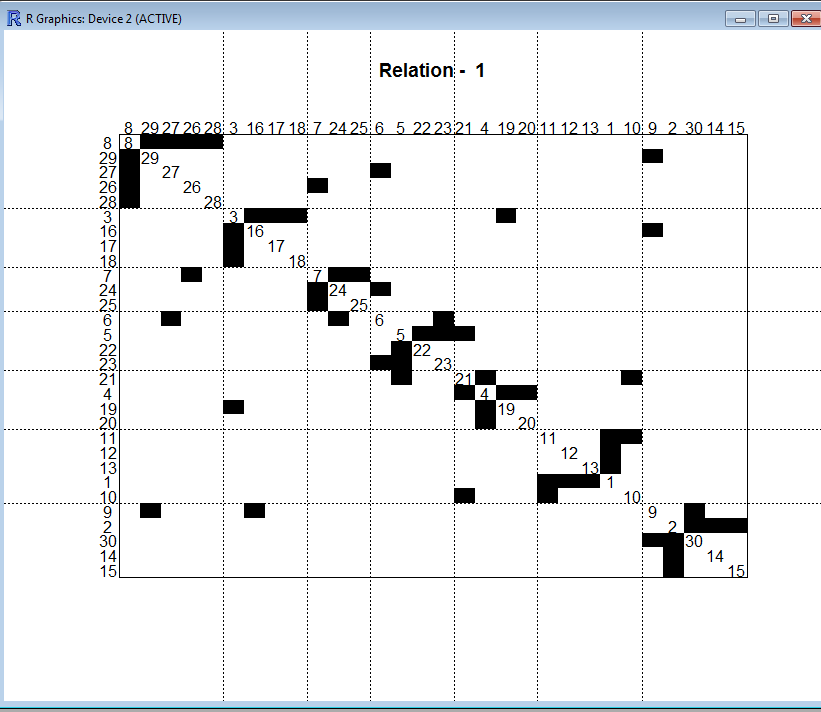


# regular equivalence from a network.

Blockmodeling

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

>plot(b)



# Practical No 7

**Create sociograms for the persons-by-persons network and the committee-bycommittee network for a given relevant problem. Create one-mode network and two-node network for the same.**

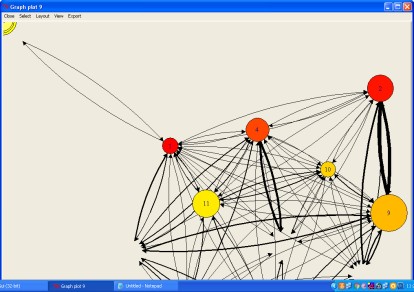
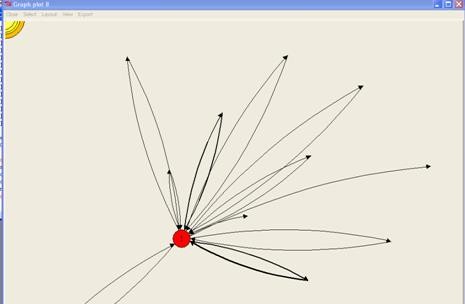
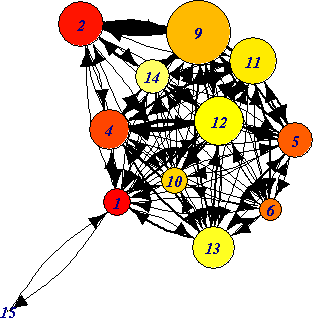
>library(Dominance)

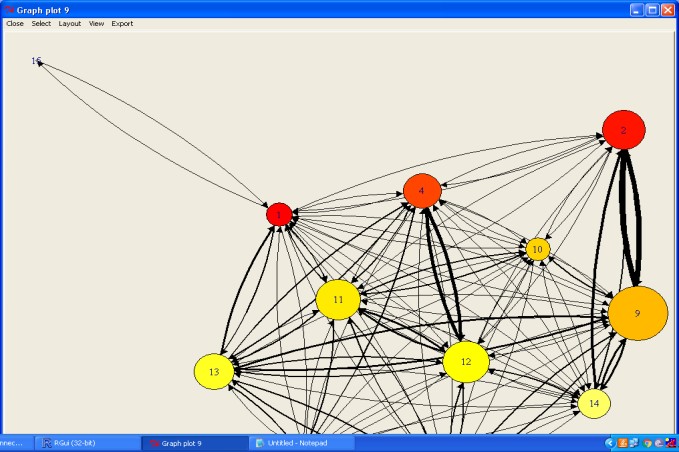
>data(data\_Network\_1)

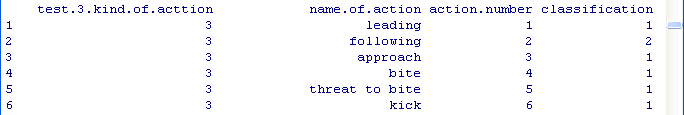
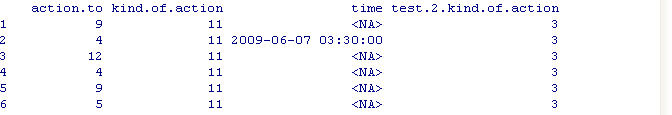
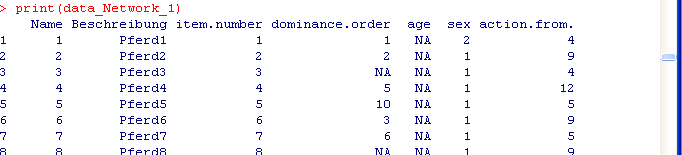
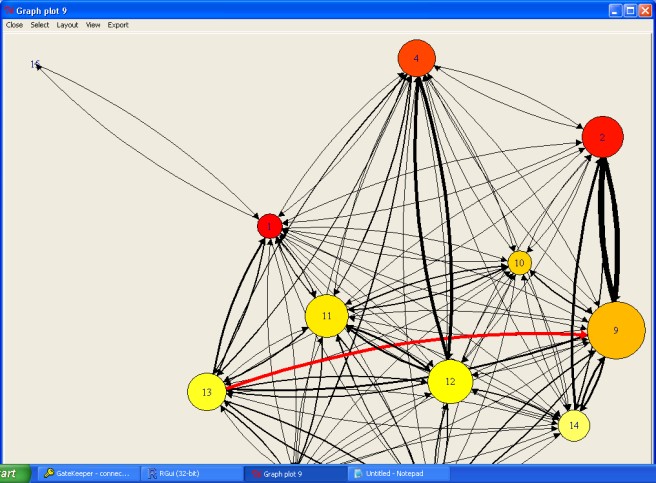
## set 1 for action you want to show

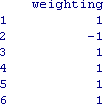
>bytes= "00111111111000000000"

>Sociogram(data\_Network\_1,bytes)









# Practical No 8

**Perform SVD analysis of a network.**

>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 9.687693e-17

$u

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [,1] | [,2] | | [,3] | [,4] | |
| [1,] -0.3333333 | | 0.4687136 | | 0.05029703 | 3.375152e-01 |
| [2,] -0.3333333 | | 0.4687136 | | 0.05029703 | -8.126230e-01 |
| [3,] -0.3333333 | | 0.4687136 | | 0.05029703 | 4.751078e-01 |
| [4,] -0.3333333 | | -0.2779153 | | 0.38076936 | 1.160461e-16 |
| [5,] -0.3333333 | | -0.2779153 | | 0.38076936 | 1.160461e-16 |
| [6,] -0.3333333 | | -0.2779153 | | 0.38076936 | 1.160461e-16 |

[7,] -0.3333333 -0.1907983 -0.43106639 -7.755807e-17

[8,] -0.3333333 -0.1907983 -0.43106639 -7.755807e-17

[9,] -0.3333333 -0.1907983 -0.43106639 -7.755807e-17

$v

|  |  |  |  |
| --- | --- | --- | --- |
| [,1] | [,2] | [,3] [,4] | |
| [1,] -0.8660254 -2.464364e-17 | | | 0.00000000 0.5 |
| [2,] -0.2886751 8.118358e-01 | | | 0.08711702 -0.5 |
| [3,] -0.2886751 -4.813634e-01 | | | 0.65951188 -0.5 |

[4,] -0.2886751 -3.304723e-01 -0.74662890 -0.5

# Practical 9

**Identify ties within the network using two-mode core periphery analysis**

**File name:** “Media-Example-NODES.csv” and “Media-Example-EDGES.csv”

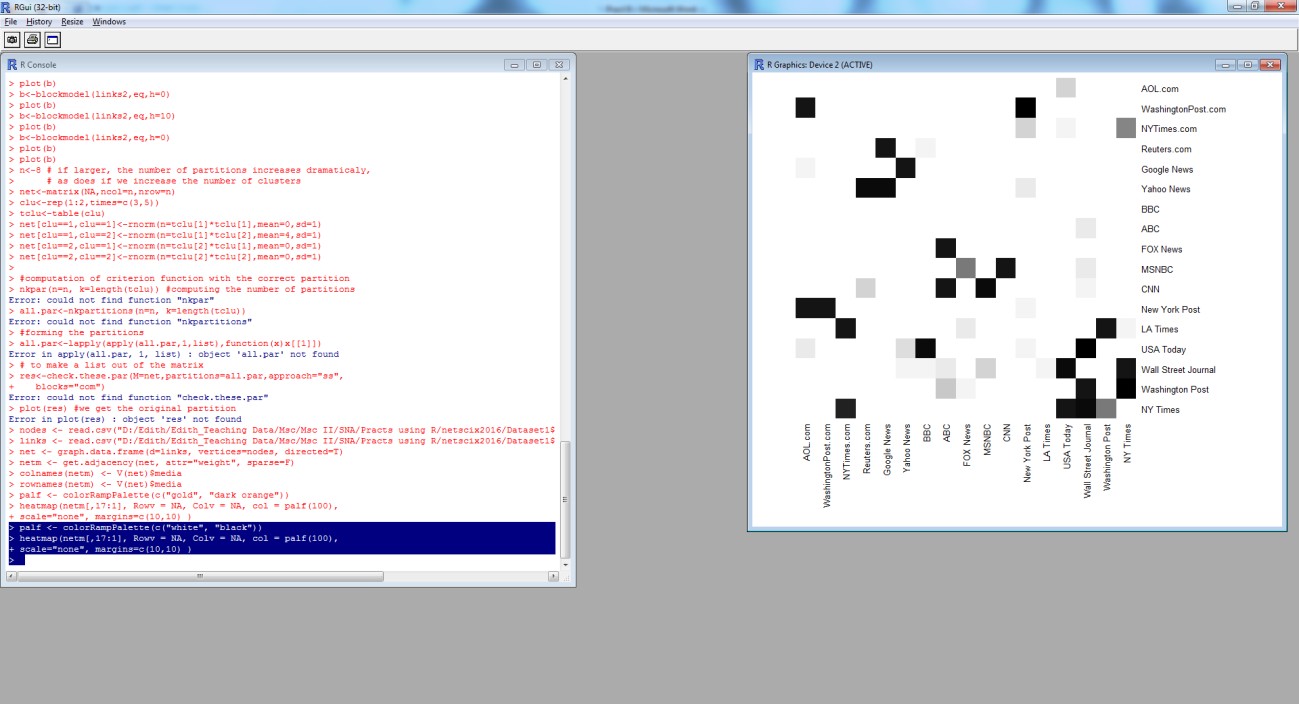
nodes<- read.csv("Dataset1-Media-Example-NODES.csv", header=T, as.is=T) links<- read.csv("Dataset1-Media-Example-EDGES.csv", header=T, as.is=T) net<- graph\_from\_data\_frame(d=links, directed=T)

netm<- get.adjacency(net, sparse=FALSE) colnames(netm) <- V(net)$media

rownames(netm) <- V(net)$media

palf<- colorRampPalette(c("white", "black"))

heatmap(netm[,17:1], Rowv = NA, Colv = NA, col = palf(100), scale="none", margins=c(10,10) )



# Practical 10

**Find “factions” in the network using two-mode faction analysis.**

install.packages(“igraphdata”)

>library(igraphdata) Warning message:

package ‘igraphdata’ was built under R version 3.0.3

>data(karate)

>set.seed(42)

>library(igraph)

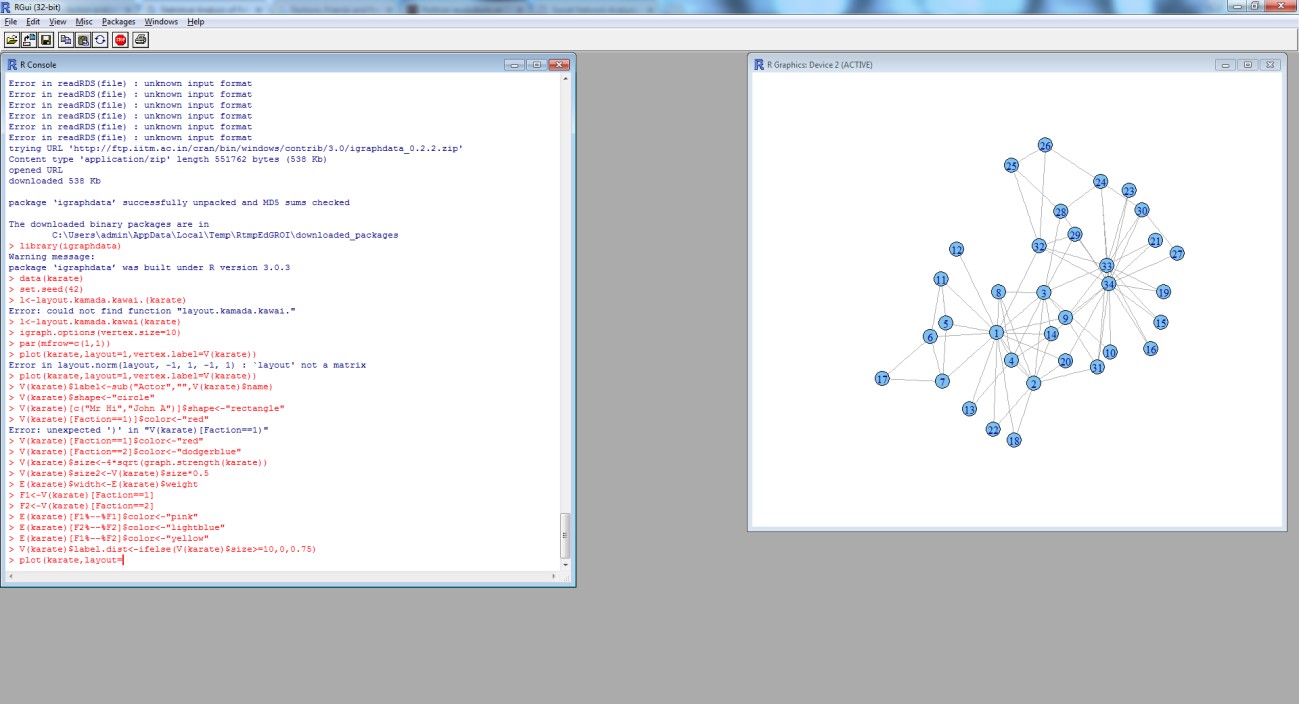
> karate <- make\_graph("Zachary")

* l<-layout.kamada.kawai(karate)

>igraph.options(vertex.size=10)

>par(mfrow=c(1,1))

>plot(karate,layout=l,vertex.label=V(karate))



>V(karate)$label<-sub("Actor","",V(karate)$name)

>V(karate)$shape<-"circle"

>V(karate)[c("MrHi","John A")]$shape<-"rectangle"

>V(karate)[Faction==1]$color<-"red"

>V(karate)[Faction==2]$color<-"dodgerblue"

>V(karate)$size<-4\*sqrt(graph.strength(karate))

>V(karate)$size2<-V(karate)$size\*0.5

>E(karate)$width<-E(karate)$weight

* F1<-V(karate)[Faction==1]
* F2<-V(karate)[Faction==2]

>E(karate)[F1%--%F1]$color<-"pink"

>E(karate)[F2%--%F2]$color<-"lightblue"

>E(karate)[F1%--%F2]$color<-"yellow"

>V(karate)$label.dist<-ifelse(V(karate)$size>=10,0,0.75)

>plot(karate,layout=l)

