Basic Graph Statistics with R

Exercise

http://www.ifi.uzh.ch/bi/teaching/fall2014/lecture/examplenode.txt
http://www.ifi.uzh.ch/bi/teaching/fall2014/lecture/exampleedge.txt

Examplenode.txt

id	rate
• 3	2
• 2	8
• 4	3
• 5	3
• 6	4
• 8	2
• 9	3
• 10	5
• 11	3

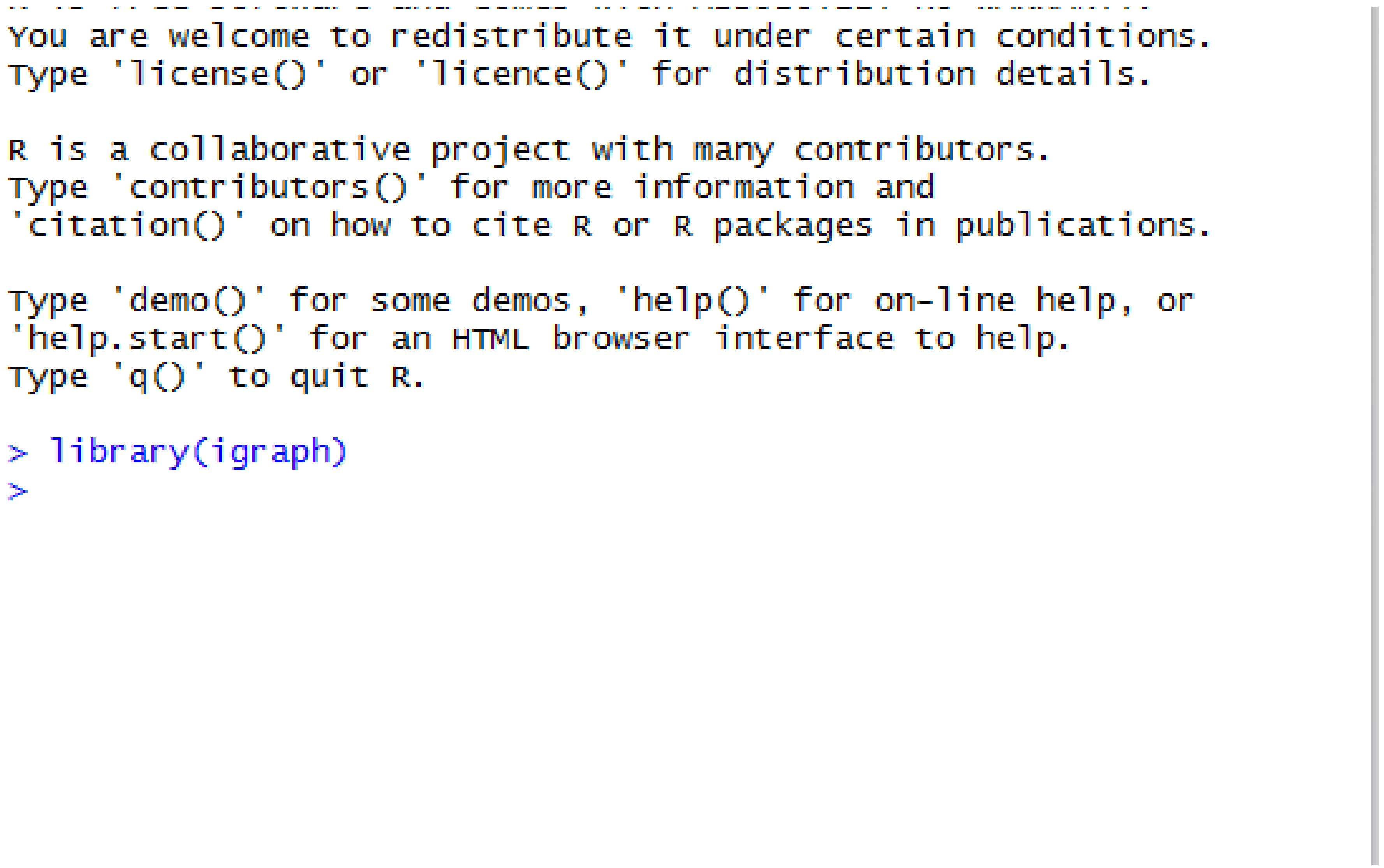
Exampleedge.txt

from	to	weight
2	3	10
2	4	3
3	4	4
4	5	9
5	7	8
7	8	4
5	8	3
3	9	1
2	10	1
8	11	3

Please calculate

- Degree
- Betweenness
- Closeness
- Visualization

1. library(igraph)



```
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> library(igraph)
>
```

2. Import data

- The <- operator sets a variable equal to something. In this case, we will set a number of basic R data structures, called "data frames," to hold the contents of the files we will open.

- read.table() is the most common R command for loading data from files in which values are in tabular format. The function loads the table into a data frame object, which is the basic data type for most operations in R. By default, R assumes that the table has no header and is delimited by any white space; these settings are fine for our purposes here.

- One handy aspect of R is that you can read in data from a URL directly by referencing the URL in the read.table() function

- If the files you want to work with are on your local machine, the easiest way to access them is to first set your working directory via the setwd() command, and then reference the files by name:

```
setwd('path/to/your_directory')
your_data_frame <- read.table('your_file_name')
```

2. Import data

- node <- read.delim('http://www.ifi.uzh.ch/bi/teaching/fall2014/lecture/examplenode.txt',header = TRUE)
- relation <- read.table('http://www.ifi.uzh.ch/bi/teaching/fall2014/lecture/exampleedge.txt',header = TRUE)
- To see the data we just loaded, it's necessary to call the variables directly
  - node
  - relation
- we can see just the top six rows via
  - head(node)
  - head(relation)

3. Loading graph

- Now we can import our data into a "graph" object using igraph's graph.data.frame() function. Coercing the data into a graph object is what allows us to perform network-analysis techniques.
  - testnet <- graph.data.frame(relation)
- By default, graph.data.frame() treats the first two columns of a data frame as an edge list and any remaining columns as edge attributes. To get a vector of edges for a specific type of tie, use the get.edge.attribute() function
  - list.edge.attributes(testnet)
  - get.edge.attribute(testnet, 'weight')
- Get the vertex of the network.
  - V(testnet)
- Get the edges of the network.
  - E(testnet)

4. ADDING VERTEX ATTRIBUTES TO A GRAPH OBJECT

- list.vertex.attributes(testnet)
- V(testnet)$name

- One way to add the attributes to your graph object is to iterate through each attribute and each vertex.
  - V(testnet)$rate = node$rate[match(V(testnet)$name, node$id)]

- V(testnet)$rate

5. Basic Graph Statistics

- degree(testnet)
- betweenness(testnet)
- closeness(testnet)
- sort(degree(testnet))

- mean(degree(testnet))
- sd(degree(testnet))

6. Visualization

- plot(testnet)
- plot.igraph(testnet)

- ?plot
- ?plot.igraph

7. Plotting cutpoints

- Cut points are called “articulation points” in igraph

- V(testnet)$color = "black“
- V(testnet)[articulation.points(testnet)]$color = "red“
- plot(testnet)