CS3907-80/CS6444-10 Big Data and Analytics
Class project 1 document

Kexin Xu, Tianyi Wang, Zhihong Liang, Ziyi Wang

June 19, 2017

1 Data preprocessing

1.1 Loading text from files

In the file "preprocessed.R", we extract text from the files. We use the gsub() and grepl() function to extract specific fields from the text. The data are collected in a data frame. Then we use na.omit() to delete the records which dont have receivers. We also use the sqldf() function in the sqldf library to group the records by message id in case there are duplicate emails. Finally, we save the result into "email_data.RData". The file "email_data.RData" contains a data frame named "newdf", which has 495554 observations.

	sender	receiver
1	richard.sanders@enron.com	rwelsh@velaw.com
2	hector.mcloughlin@enron.com	susan.wimberley@enron.com
3	tana.jones@enron.com	alan.aronowitz@enron.com, harry.collins@enron.com
4	shirley.crenshaw@enron.com	vince.kaminski@enron.com, stinson.gibner@enron.c...
5	patrice.mims@enron.com	mims@enron.com, mimsc@ops.org
6	amanda.rybarski@enron.com	edith.cross@enron.com, mike.curry@enron.com, leo...
7	don.baughman@enron.com	kayne.coulter@enron.com, john.kinser@enron.com, ...
8	mark.taylor@enron.com	louise.kitchen@enron.com
9	john.kiani-aslani@enron.com	gerald.nemec@enron.com
10	barry.tycholiz@enron.com	f..calger@enron.com
11	richard.shapiro@enron.com	tom.briggs@enron.com
12	scott.griffin@enron.com	grahamk@usfilter.com, fieldst@usfilter.com
13	m..love@enron.com	christy.lobusch@enron.com
14	peter.makkai@enron.com	joe.stepenovitch@enron.com
15	jeff.dasovich@enron.com	james.steffes@enron.com
16	d..steffes@enron.com	jeff.dasovich@enron.com

Showing 1 to 17 of 495,554 entries

1.2 Deleting the less important data

In the file "simplify.R", we separate the receivers and delete the less important records. Some email may have multiple receivers, therefore we split them using the strsplit() function and create rows which only have one receiver. We use count() function in the plyr library to count the frequency of senders and receivers, and then we delete the senders and receivers who are below the average of frequency. Finally, we save the result into "processed.RData". The file "processed.RData" contains a data frame named "newdf3", which has 1456222 observations.

	sender	receiver
3	tana.jones@enron.com	alan.aronowitz@enron.com
4	tana.jones@enron.com	harry.collins@enron.com
5	shirley.crenshaw@enron.com	vince.kaminski@enron.com
6	shirley.crenshaw@enron.com	stinson.gibner@enron.com
7	shirley.crenshaw@enron.com	pinnamaneni.krishnarao@enron.com
8	shirley.crenshaw@enron.com	vasant.shanbhogue@enron.com
9	shirley.crenshaw@enron.com	mike.roberts@enron.com
10	shirley.crenshaw@enron.com	joseph.hrgovcic@enron.com
11	shirley.crenshaw@enron.com	tanya.tamarchenko@enron.com
12	shirley.crenshaw@enron.com	zimin.lu@enron.com
13	shirley.crenshaw@enron.com	martin.lin@enron.com
14	shirley.crenshaw@enron.com	maureen.raymond@enron.com
15	shirley.crenshaw@enron.com	osman.sezgen@enron.com
18	shirley.crenshaw@enron.com	alex.huang@enron.com
19	shirley.crenshaw@enron.com	kevin.moore@enron.com
20	shirley.crenshaw@enron.com	william.smith@enron.com

Showing 1 to 17 of 1,456,222 entries

2 Graph generating

2.1 Getting the adjacency matrix

At first, the functions as.matrix() and graph.edgelist(matrix, directed = TRUE) were implemented to acquire the edge list of the data set. Then, get.adjacency() was used to convert the edge list to adjacency matrix. Then, all the relationship information is stored in the matrix consisting of connection frequencies.

```
tana.jones@enron.com             11 809 855    .    .   2    .    .    .    .
.
alan.aronowitz@enron.com         28   .  13    .   3    .    .    .    .    .
.
harry.collins@enron.com          22   .   .    .   .    .    .    .    .    .
.
shirley.crenshaw@enron.com        .   .   .    . 364 188 163 180 144 124
164
vince.kaminski@enron.com          .   4   . 1246 164 501 146 208 225  62
117
stinson.gibner@enron.com          .   .   .   31 184   .   9   3   3   3
9
pinnamaneni.krishnarao@enron.com  .   .   .   21  66   5   4   7   .   1
```

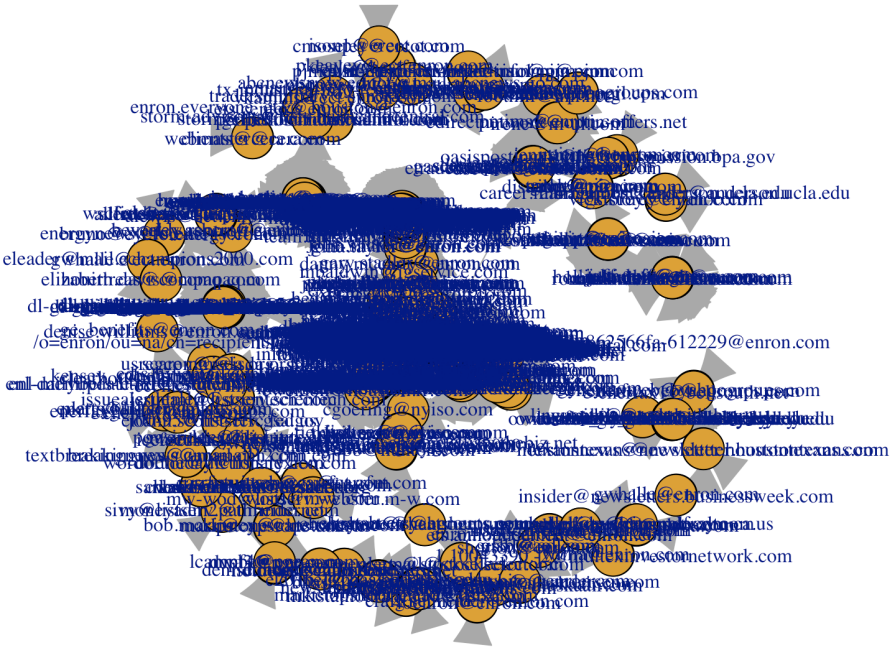
2.2 Getting the graph of our data

Based on the adjacency matrix, graph.adjacency(matrix, mode = "directed", weighted = TRUE) was implemented to convert the adjacency matrix to network graph.

```
IGRAPH DNW- 2631 12086 --
+ attr: name (v/c), weight (e/n)
+ edges (vertex names):
 [1] tana.jones@enron.com->alan.aronowitz@enron.com
 [2] tana.jones@enron.com->harry.collins@enron.com
 [3] tana.jones@enron.com->souad.mahmassani@enron.com
 [4] tana.jones@enron.com->dutch.quigley@enron.com
 [5] tana.jones@enron.com->mark.taylor@enron.com
 [6] tana.jones@enron.com->louise.kitchen@enron.com
 [7] tana.jones@enron.com->brent.hendry@enron.com
 [8] tana.jones@enron.com->david.forster@enron.com
+ ... omitted several edges
>
```

2.3 Simplifying the graph

At first, simplify(Graph, remove.multiple = TRUE, remove.loops = TRUE, edge.attr.comb = igraph_opt("edge.attr.comb")) was used to cut all the replications to acquire a relatively simple version of graph. Then E(Graph)$weight and mean() were implemented to calculate the average connection frequency. After that, delete.edges(newGraph, which(E(newGraph)$weight<=ave)) was used to delete all the edges below to average frequency in order to eliminate the minor relationships and simplify the graph again. At last, delete.vertices(Graph, which(degree(newGraph)<1)) was used to delete isolated nodes to increase the readability of graph and plot(Graph) was used to plot the graph out.



3 Graph analyzing

3.1 Central person

In order to determine the central person, we need to first take into consideration that Enron was a very large organization. There are many different definitions for the term "central" when it comes to a communication network. Therefore we took 3 different approaches:

1) Select the person who has the top degree centrality. Degree centrality measures how many direct connections a person has to other people within the network. In this way we can find out the individual who has the most connections. In other words, such individual should hold most knowledge of the organization.

```
> #degree centrality for business network
> deg <- degree(finalGraph,loops=FALSE)
> head(sort(deg,decreasing=TRUE))
    david.forster@enron.com      jeff.dasovich@enron.com         tana.jones@enron.com
                        292                          236                          222
veronica.espinoza@enron.com       outlook.team@enron.com        steven.kean@enron.com
                        211                          191                          170
```

2) Identify the individual who has the highest betweenness centrality. Here we are measuring the number of times an individual lies on the shortest commu-

nication chain between other people. Higher betweenness centrality indicates higher authority. Since such individual often controls collaboration between people or clusters.

```
> #betweenness centrality
> head(sort(betweenness(finalGraph),decreasing = TRUE))
 jeff.dasovich@enron.com     sally.beck@enron.com david.delainey@enron.com shelley.corman@enron.com
               173859.7                152411.4                139526.5                131101.9
    tana.jones@enron.com vince.kaminski@enron.com
               124951.9                120151.2
```

3) Finding the node with the highest closeness centrality. This is the most straightforward method. It calculates the shortest path between all people, then assign a value to each person based on its sum of shortest path. It can give us the individual who can influence Enron most quickly.

```
> head(sort(closeness(finalGraph),decreasing = TRUE))
 outlook.team@enron.com david.forster@enron.com     simone.la@enron.com    liz.taylor@enron.com
          5.360298e-07            5.352310e-07            5.298729e-07            5.298106e-07
 julie.clyatt@enron.com fraisy.george@enron.com
          5.297140e-07            5.291089e-07
```

Conclusion Even though different measurement gives us different results, some results are shared. By combining the results and take the top occurrences, we determined that the central people of Eron from this data set are:

1. david.forster@enron.com
2. jeff.dasovich@enron.com

We decided to exclude the team email from our results since it is irrelevant to this problem.

3.2 Longest path

In order to determine the longest path, we used the diameter() function in igraph package on our previously simplified and established graph. In this dataset, the longest path has 2206 nodes, and is between node enron.announcements@enron.com and all.worldwide@enron.com.

```
> diameter(finalGraph)
[1] 2206
> farthest.nodes(finalGraph)
$vertices
+ 2/2250 vertices, named:
[1] enron.announcements@enron.com all.worldwide@enron.com

$distance
[1] 2206
```

3.3 Largest clique

By using the built-in largest_cliques() function, we found out that the largest clique in this network is a 17-node one. The implementation and result is:

```
> #calculate the largest clique in the network
> largest_cliques(finalGraph)
[[1]]
+ 17/2250 vertices, named:
 [1] keegan.farrell@enron.com    holly.keiser@enron.com      mary.cook@enron.com
 [4] sara.shackleton@enron.com   taffy.milligan@enron.com    janette.elbertson@enron.com
 [7] mark.taylor@enron.com       carol.clair@enron.com       becky.spencer@enron.com
[10] susan.bailey@enron.com      stephanie.panus@enron.com   outlook.team@enron.com
[13] tana.jones@enron.com        samantha.boyd@enron.com     frank.sayre@enron.com
[16] cheryl.nelson@enron.com     marie.heard@enron.com
```

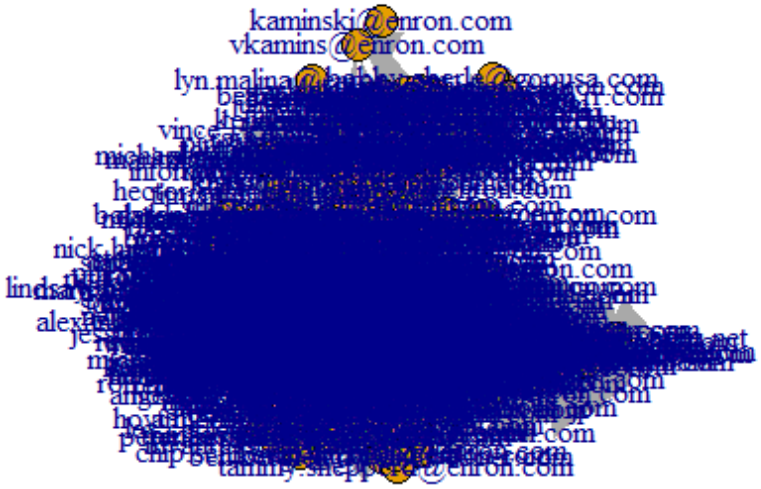
3.4 Ego

We've tried to make and plot an ego network of a given vertex in this larger network, Eron. At first, we used the function "make_ego_graph" from igraph package to build graphs for each vertex. We set the order 3 which means the graph shows all vertices that are within 3 node lengths from our chosen vertex.

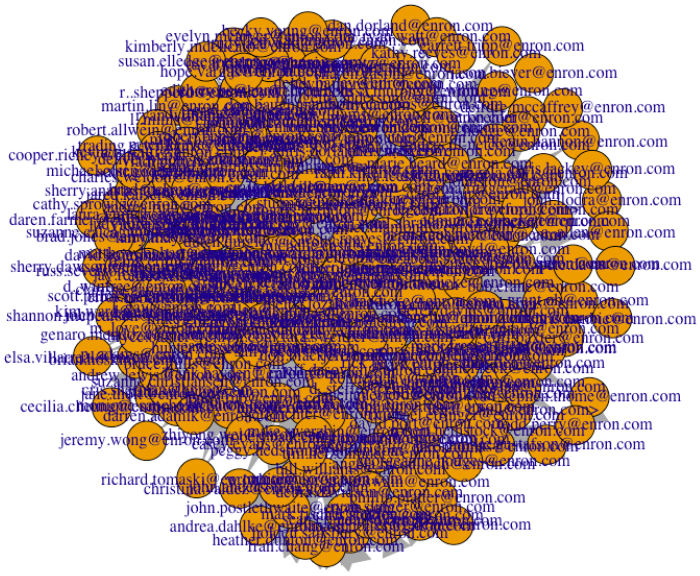
```
graph3<- make_ego_graph(graph = finalGraph, order = 3, nodes = V(finalGraph), mode = c("all", "out", "in"), mindist = 0)
```

Since the result is presented list format, we can plot different ego graphs from the list by choosing different item in the list. For example:

```
> plot(graph3[[1]])
> plot(graph3[[11]])
```



We found it yields a list of graphs which are more than 1000. In order to just plot the graph for a specific vertex, we added the attribute name="" . For example, here we are looking at the the ego graph of david.forster@enron.com , who we previously determined as the central person of Enron:



```
> ego_graph3 <-make_ego_graph(graph = finalGraph, order =1, node = V(finalGraph)$name == "david.forster@en
ron.com", mode = c("all","out","in"), mindist=0)
> plot(ego_graph3[[1]])
```

Also, we can change the mode to specify the direction of the edges. For "out" only the outgoing edges are followed, so all vertices reachable from the source vertex in at most order steps are counted. For "in" all vertices from which the source vertex is reachable in at most order steps are counted. "all" ignores the direction of the edges.

3.5 Power centrality

We simply used power_centrality() function from the igraph package to determine power centrality of this dataset. In this case the power centrality is angela.white@enron.com. We experienced some problems when trying to use this function on this relevantly larger dataset, by playing around with exponent, we were able to obtain results.

```
> power<-power_centrality(finalGraph,loops=FALSE,exponent = 0.5, rescale = 1, tol = 1e-07, sparse = TRUE)
> head(sort(power,decreasing = TRUE))
 angela.white@enron.com     dennis.lee@enron.com      kenny.ha@enron.com craig.buehler@enron.com
             0.17630498              0.14578548              0.12476524              0.11278149
   joe.parks@enron.com airam.arteaga@enron.com
             0.10211805              0.06337946
```

4 List of R functions

1. In data preprocessing section: list.files(), lapply(), gsub(), grepl()/grep(), is.na(), paste(), data.frame(), na.omit(), sqldf(), save(), load(), count(), mean(), strsplit(), as.character(), data.frame(), rep(), sapply()

2. In graph generating section: as.matrix(), graph.edgelist(), get.adjacency(), graph.adjacency(), simplify(), E(Graph)$weight, mean(), delete.edges(), delete.vertices(), plot()

3. In graph analyzing section: degree(), betweenness(), closeness(), diameter(), farthest.nodes(), largest_cliques(), power_centrality(), plot(), head(), sort(), make_ego_graph()

5 What we learned about the data

1. The original data contains lots of redundant information, therefore data processing is necessary for further analysis.

2. Processing data is not easy as it sounds. In this project we have processed our original data multiple times to reach a relevantly simpler dataset for us to deal with.

3. There is significant amount of information we could learn from this seemly simple email data of Enron. From our analysis of centrality, we found out that the two central people are the Vice President of the company, and a government relations official, which is quite odd considering the fact that normally you would not expect to receive large amount of emails from the government relations official. Even without the knowledge of Enron, we could say that this company is experiencing some sort of problem.

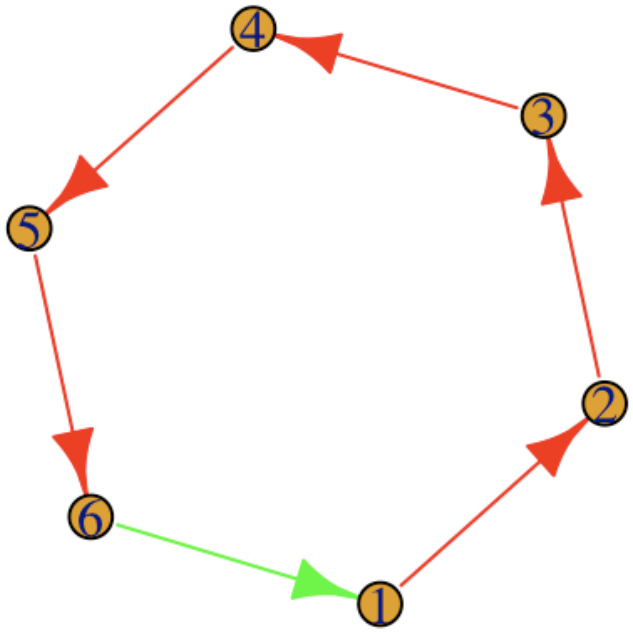
4. The communication is quite efficient in Enron by making use of the company email. The average communication chain in this dataset is about 4.17.

5. Due to the limits of hardware and time, the simplification of data is necessary in most scenarios. Thus, it is important to cut off the minor information in the data set. After several times of simplification, the most significant data will show up and a more accurate result can be easily acquired.

6 Explore the igraph package

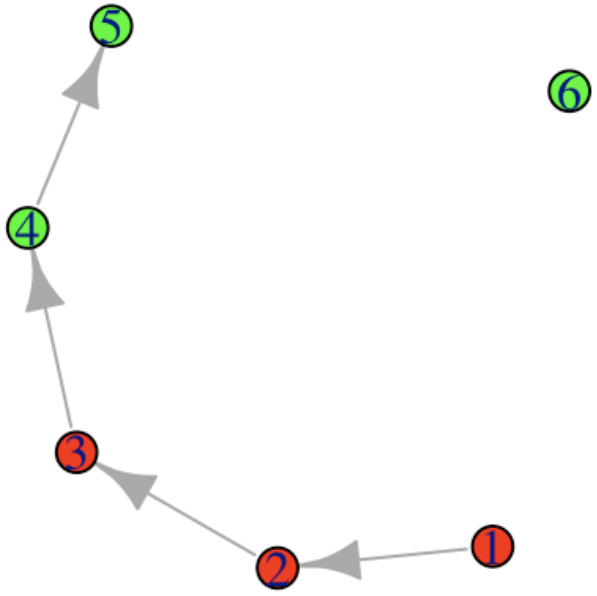
6.1 Add an edge

add_edges(graph, edges, ..., attr = list())



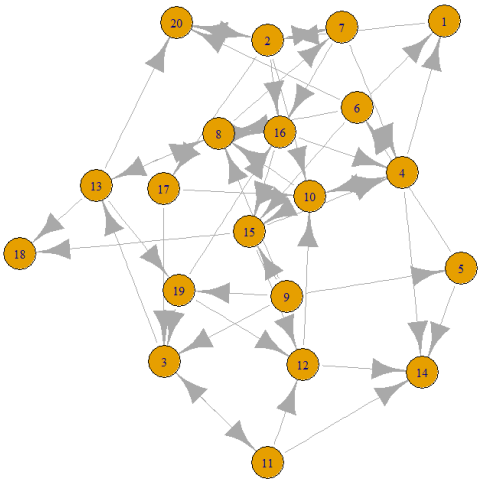
6.2 Add vertices

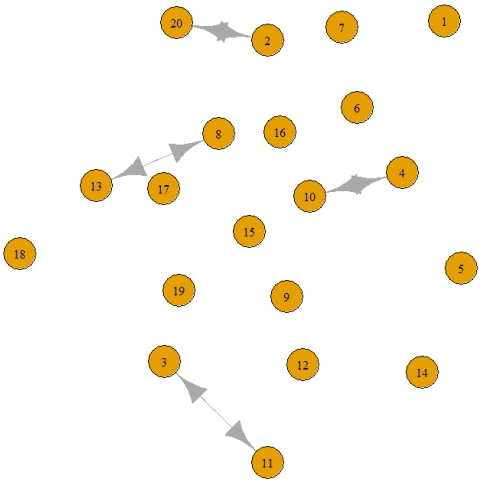
add_vertices(graph, nv, ..., attr = list())



6.3 Find mutual edges

murg <- delete.edges(rg, which(!is.mutual(rg)))


20
2
7
1
6
8
16
4
13
17
10
15
18
5
19
9
3
12
14
11


20
2
7
1
6
8
16
13
17
4
10
15
18
5
19
9
3
12
14
11

6.4 BFS search of a graph

print(graph.bfs(rg, V(rg)[1], father = T, pred = T, succ = T))

```
> print(graph.bfs(rg, V(rg)[1], father = T, pred = T, succ = T))
$root
[1] 1

$neimode
[1] "out"

$order
+ 20/20 vertices:
 [1]  1  2  7 10 16 17 20  4  8 15  3 14 13 12 18 11 19  5  6  9

$rank
NULL

$father
+ 20/20 vertices:
 [1] NA  1 17  7 NA  5  2 10 NA  2  3 15  8  4 10  2  2 15 13  2

$pred
+ 20/20 vertices:
 [1] NA  1 15 20 NA  5  2  4 NA  7 18 13 14  3  8 10 16 12 11 17

$succ
+ 20/20 vertices:
 [1]  2  7 14  8  6 NA 10 15 NA 16 19 18 12 13  3 17 20 11 NA  4

$dist
NULL
```

6.5 DFS search of a graph

print(graph.dfs(rg, V(rg)[1], father = T))

```
> print(graph.dfs(rg, V(rg)[1], father = T))
$root
[1] 0

$neimode
[1] "out"

$order
+ 20/20 vertices:
 [1]  1  2  7  4 10  8 13 18 19  3 11 12 14 16 15 20 17  5  6  9

$order.out
NULL

$father
+ 20/20 vertices:
 [1] NA  1 19  7 NA  5  2 10 NA  4  3 11  8 12 16 19  2 13 13 13

$dist
NULL
```

6.6 Average path length

The function "average.path.length()" was used to explore our dataset in order to gain better understanding of the graph we were dealing with.

```
> #Average path length
> average.path.length(finalGraph,directed=TRUE,unconnected=TRUE)
[1] 4.170333
```

6.7 Total number of nodes

By summing up the number of vertices within a graph, this function gives us the total number of nodes. We used this function when inspecting our simplified graph.

```
> #Total number of nodes
> vcount(finalGraph)
[1] 2250
```

6.8 Get shortest path

shortest.paths()

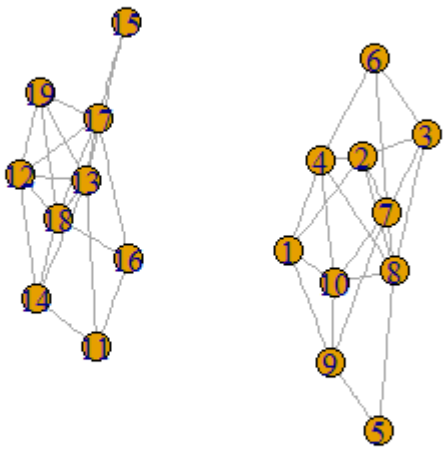
```
> g<-graph.ring(10)
> shortest.paths(g)
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]    0    1    2    3    4    5    4    3    2     1
 [2,]    1    0    1    2    3    4    5    4    3     2
 [3,]    2    1    0    1    2    3    4    5    4     3
 [4,]    3    2    1    0    1    2    3    4    5     4
 [5,]    4    3    2    1    0    1    2    3    4     5
 [6,]    5    4    3    2    1    0    1    2    3     4
 [7,]    4    5    4    3    2    1    0    1    2     3
 [8,]    3    4    5    4    3    2    1    0    1     2
 [9,]    2    3    4    5    4    3    2    1    0     1
[10,]    1    2    3    4    5    4    3    2    1     0
```

```
> get.shortest.paths(g,5)
$vpath
$vpath[[1]]
+ 5/10 vertices:
[1] 5 4 3 2 1

$vpath[[2]]
+ 4/10 vertices:
[1] 5 4 3 2

$vpath[[3]]
+ 3/10 vertices:
[1] 5 4 3

$vpath[[4]]
+ 2/10 vertices:
[1] 5 4

$vpath[[5]]
+ 1/10 vertex:
[1] 5

$vpath[[6]]
+ 2/10 vertices:
[1] 5 6

$vpath[[7]]
+ 3/10 vertices:
[1] 5 6 7

$vpath[[8]]
+ 4/10 vertices:
[1] 5 6 7 8

$vpath[[9]]
+ 5/10 vertices:
[1] 5 6 7 8 9

$vpath[[10]]
+ 6/10 vertices:
[1]  5  4  3  2  1 10
```

6.9 Generate a random graph

generate a random graph with a fix number of vertices and every possible edge is created with the same constant probability.

```
> g<-sample_gnp(10, 5/10) %du% sample_gnp(9, 5/9)
> plot(g)
```



6.10 Finding communities based on propagating labels

detect the community structure in networks and label the vertices with unique labels.

```
> g <- add_edges(g, c(1, 12))
> cluster_label_prop(g)
IGRAPH clustering label propagation, groups: 2, mod: 0.48
+ groups:
  $`1`
   [1]  1  2  3  4  5  6  7  8  9 10

  $`2`
  [1] 11 12 13 14 15 16 17 18 19
```