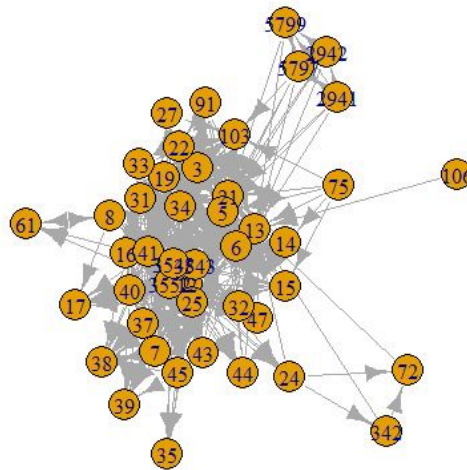


### Simplifying the graph:

We simplified our graph by using the `simplify` function in `igraph` removing loop and multiples, then we looked at the distribution of degrees using `degree_distribution`. Based on what we saw from the distribution data we deleted vertices which had degrees less than or equal to 200. this is what our network looked like after:



## Functions applied from lecture slides:

### 1. geodist

description: uses a BFS to find the number and lengths of geodesics between all nodes of dat. Where geodesics do not exist, the value in inf.replace is substituted for the distance in question.

output:

```
sgdist
[1,] [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14] [,15] [,16] [,17] [,18] [,19] [,20]
[2,] Inf 0 Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf
[3,] 1 1 0 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 Inf 1 Inf
[4,] Inf Inf Inf 0 Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf
[5,] 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 Inf 1 1
[6,] Inf Inf Inf Inf Inf 0 Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf
[7,] Inf Inf Inf Inf Inf Inf 0 Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf
[8,] Inf Inf Inf Inf Inf Inf Inf 0 Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf
[9,] Inf 2 Inf Inf Inf Inf 3 1 0 1 3 2 Inf 3 2 Inf Inf Inf 3 2
[10,] Inf 1 Inf Inf Inf Inf 2 1 1 0 2 1 Inf 2 1 Inf Inf Inf 2 1
[11,] Inf 3 Inf Inf Inf Inf 2 3 3 2 0 2 Inf 2 1 Inf Inf Inf 1 2
[12,] Inf 2 Inf Inf Inf Inf 3 2 2 1 1 0 Inf 3 2 Inf Inf Inf 2 1
[13,] Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf 0 Inf Inf Inf Inf Inf Inf Inf
[14,] Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf 0 Inf Inf Inf Inf Inf Inf Inf
[15,] Inf Inf Inf Inf Inf Inf 1 Inf Inf Inf Inf Inf 1 0 Inf Inf Inf Inf Inf Inf
[16,] 2 2 1 2 2 1 1 1 1 2 1 1 1 1 2 0 2 Inf 1 1
[17,] Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf 0 Inf Inf Inf
[18,] 2 2 1 2 2 2 2 2 2 2 2 2 3 1 2 2 2 0 2 2
[19,] Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf 0 Inf
[20,] Inf 3 Inf Inf Inf Inf 4 3 3 2 2 1 Inf 4 3 Inf Inf Inf 3 0
[21,] Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf Inf
[22,] Inf 2 Inf Inf Inf Inf 3 2 2 1 1 1 Inf 3 2 Inf Inf Inf 2 1
[23,] Inf Inf Inf Inf Inf Inf Inf 1 1 1 1 Inf Inf Inf 2 Inf Inf Inf Inf Inf
[24,] Inf Inf Inf Inf Inf Inf Inf 1 1 1 1 Inf Inf Inf 2 Inf Inf Inf Inf Inf
[25,] Inf Inf Inf Inf Inf Inf Inf 1 1 1 1 Inf Inf Inf 2 Inf Inf Inf Inf Inf
[26,] Inf Inf Inf Inf Inf Inf Inf 1 1 1 1 Inf Inf Inf 2 Inf Inf Inf Inf Inf
[27,] Inf Inf Inf Inf Inf Inf Inf 1 1 1 1 Inf Inf Inf 2 Inf Inf Inf Inf Inf
[28,] Inf Inf Inf Inf Inf Inf Inf 1 1 1 1 Inf Inf Inf 2 Inf Inf Inf Inf Inf
[29,] Inf Inf Inf Inf Inf Inf Inf 1 1 1 1 Inf Inf Inf 2 Inf Inf Inf Inf Inf
[30,] Inf Inf Inf Inf Inf Inf Inf 1 1 1 1 Inf Inf Inf 2 Inf Inf Inf Inf Inf
[31,] Inf Inf Inf Inf Inf Inf Inf 1 1 1 1 Inf Inf Inf 2 Inf Inf Inf Inf Inf
[32,] Inf Inf Inf Inf Inf Inf Inf 1 1 1 1 Inf Inf Inf 2 Inf Inf Inf Inf Inf
[33,] Inf Inf Inf Inf Inf Inf Inf 1 1 1 1 Inf Inf Inf 2 Inf Inf Inf Inf Inf
[34,] Inf Inf Inf Inf Inf Inf Inf 1 1 1 1 Inf Inf Inf 2 Inf Inf Inf Inf Inf
[35,] Inf Inf Inf Inf Inf Inf Inf 1 1 1 1 Inf Inf Inf 2 Inf Inf Inf Inf Inf
[36,] Inf Inf Inf Inf Inf Inf Inf 1 1 1 1 Inf Inf Inf 2 Inf Inf Inf Inf Inf
[37,] Inf Inf Inf Inf Inf Inf Inf 1 1 1 1 Inf Inf Inf 2 Inf Inf Inf Inf Inf
[38,] Inf Inf Inf Inf Inf Inf Inf 1 1 1 1 Inf Inf Inf 2 Inf Inf Inf Inf Inf
[39,] Inf Inf Inf Inf Inf Inf Inf 1 1 1 1 Inf Inf Inf 2 Inf Inf Inf Inf Inf
```

theres a lot of inf so there are not that many geodesics within in our network

### 2. gden

description: computes the density of the graphs indicated by g in collection dat, adjusting for the type of graph in question.

output:

```
> gden(g5_adj)
[1] 0.1737374
```

### 3. connectedness

description: takes one or more graphs (dat) and returns the Krackhardt connectedness scores for the graphs selected by g

output:

```
> connectedness(g5_adj)
[1] 1
```

the output is one so the graph is highly connected

### 4. is.connected

is.connected determines whether the elements of g are connected under the definition specified in connected

```
> is.connected(g5_adj)
Node 1, Reach 6, Total 6
Node 2, Reach 10, Total 16
Node 3, Reach 35, Total 51
Node 4, Reach 6, Total 57
Node 5, Reach 35, Total 92
Node 6, Reach 6, Total 98
Node 7, Reach 6, Total 104
Node 8, Reach 6, Total 110
Node 9, Reach 18, Total 128
Node 10, Reach 18, Total 146
Node 11, Reach 18, Total 164
Node 12, Reach 18, Total 182
Node 13, Reach 6, Total 188
Node 14, Reach 6, Total 194
Node 15, Reach 8, Total 202
Node 16, Reach 35, Total 237
Node 17, Reach 1, Total 238
Node 18, Reach 36, Total 274
Node 19, Reach 6, Total 280
Node 20, Reach 18, Total 298
Node 21, Reach 1, Total 299
Node 22, Reach 18, Total 317
Node 23, Reach 35, Total 352
Node 24, Reach 1, Total 353
```

---

### 5. `as_adjacency_matrix`

This function produces a matrix of size  $n \times n$ , where  $n$  is the number of nodes in the graph. 1s denote the existence of an edge between the corresponding nodes.

```
Console Terminal x Jobs x
~/Desktop/project 1/program/ ↗
... omitted several edges
> adjMat<-as_adjacency_matrix(adjMat)
> adjMat
45 x 45 sparse Matrix of class "dgCMatrix"
[[ suppressing 45 column names '47', '31', '12' ... ]]
[[ suppressing 45 column names '47', '31', '12' ... ]]

47 . . . . . 1 1 1 . . . . .
31 . . . . . 1 . . . . . 1 1 1 . . . . . 1
12 1 1 . 1 1 1 1 1 1 1 1 . 1 1 1 1 1 1 1 1 1 . . . 1 1 . 1 . . . . . 1
44 . . . . . 1 1 1 . . . . .
25 1 1 1 1 . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . 1 1 . 1 . . . . .
32 . . . . . 1 1 1 . . . . .
3 . . . . . 1 1 1 . . . . .
19 . . . . . 1 1 1 . . . . . 1
33 . . . . . 1 . 1 . . . . . 1 1 1 . . . . . 1
16 . 1 . . . . 1 1 . . 1 . . . . 1 . . . . 1 1 1 . . . . .
43 . . . . . 1 . . . . 1 . . . . 1 1 1 . . . . .

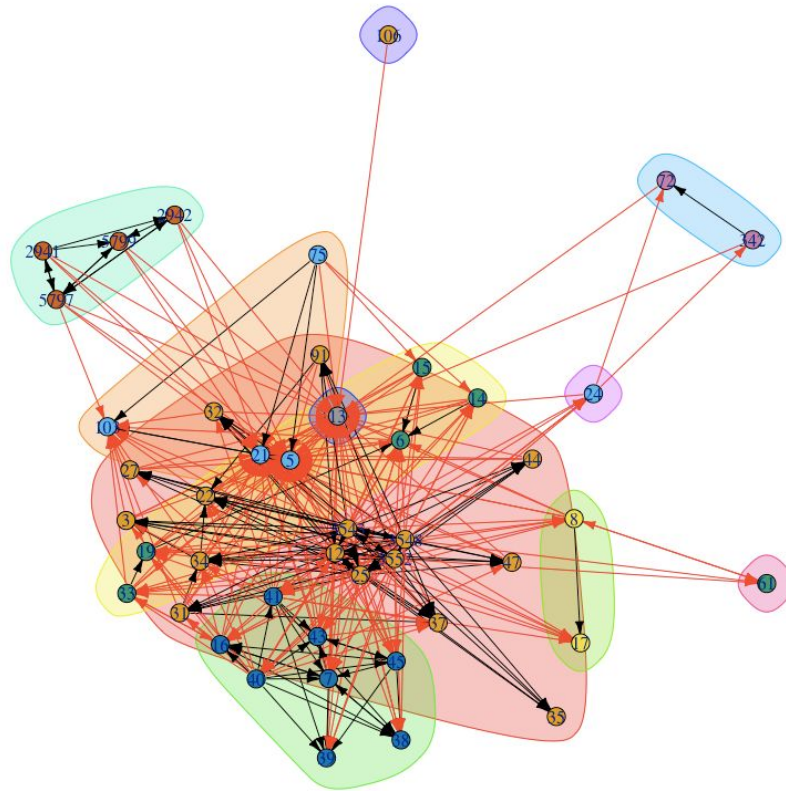
.....
.....suppressing 23 rows in show(); maybe adjust 'options(max.print= *, width = *)'
.....
[[ suppressing 45 column names '47', '31', '12' ... ]]

24 . . . . . 1 . . . 1 1 . . . . .
61 . . . . . 1 . . . . . 1 . . . . .
8 . . . . . 1 . . . . . 1 1 1 . . . . . 1
3543 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . 1 1 . 1 1 . . . . . 1
3552 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . 1 1 1 1 1 . 1 . . . .
3548 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . 1 1 1 1 1 1 . . . . .
5797 . . . . . 1 1 1 . . . . . 1 1 1 1 . . . . .
5799 . . . . . 1 1 1 . . . . . 1 . . . .
2942 . . . . . 1 1 1 . . . . . 1 1 . . .
2941 . . . . . 1 1 1 . . . . . 1 1 1 . . .
103 . . . . .
> |
```

## 6. community(graph)

The community function enables the user to find densely connected subgraphs. This lets one determine how the data is structured.

```
> wcg5<-walktrap.community(g5)
> plot(wcg5,g5,vertex.size=5, edge.arrow.size=0.25, layout=layout.fruchterman.reingold)
```



## 7. alpha centrality

```
> alphag5<-alpha centrality(g5, alpha=2)
> View(alphag5)
```

```
> alphag5
```

47	31	12	44	25	32	3	19
1.23529412	-2.13368984	0.56862745	0.41176471	0.13725490	2.88235294	-8.64705882	1.01069519
33	16	43	7	27	22	34	41
0.33689840	-1.68449198	0.11229947	-3.25668449	1.74509804	-10.64705882	-6.17647059	1.23529412
35	91	37	38	39	45	40	17
0.41176471	-1.00000000	-0.33689840	-3.25668449	0.11229947	1.45989305	0.41176471	3.05228758
15	14	21	13	5	75	106	342
2.41176471	2.41176471	0.16526611	5.96918768	0.16526611	1.00000000	1.00000000	1.82352941
72	6	24	61	8	3543	3552	3548
5.47058824	12.58068076	0.41176471	-0.16339869	0.08496732	-0.33333333	-0.33333333	-0.33333333
5797	5799	2942	2941	103			
-0.52941176	-0.52941176	-0.17647059	-0.05882353	-15.02427638			

## 8. edge.disjoint.paths

This calculates the number of paths which do not contain any common edges. This number is the minimum number of edges which have to be removed to get rid of all directed paths between a given source and a target.

```
> edge.disjoint.paths(g6, 7, 39)
[1] 0
> edge.disjoint.paths(g6, 5, 8)
[1] 6
> edge.disjoint.paths(g6, 22, 34)
[1] 2
```

---

## 9. page\_rank

This function calculates the Google PageRank of all the vertices in the graph. PageRank measures the importance of web keyword search results.

```
> page_rank(g6)
$vector
      47      31      12      44      25      32      3
0.007225003 0.008340646 0.008462457 0.006860537 0.006671511 0.007244365 0.009177787
      19      33      16      43      7      27      22
0.009960441 0.008724474 0.011812690 0.010960551 0.019921875 0.007004595 0.010271183
      34      41      35      91      37      38      39
0.011074982 0.007225003 0.006860537 0.006431741 0.010343167 0.012076194 0.010960551
      45      40      17      15      14      21      13
0.008777748 0.006860537 0.009208971 0.007706889 0.007706889 0.159211375 0.157402908
      5      75      106      342      72      6      24
0.168512383 0.005974253 0.005974253 0.007918072 0.011283252 0.049602728 0.006860537
      61      8      3543      3552      3548      5797      5799
0.008243851 0.013867810 0.006279245 0.006279245 0.006279245 0.010463091 0.009677203
      2942      2941      103
0.008271114 0.007244771 0.112783339

$value
[1] 1

$options
NULL

> |
```

---

## 10. vertex\_attr

“Vertex\_attr” classifies all the vertices in a graph by the value or name given. The name of a specific attribute can be specified in the function, and if not, all the vertex attributes are shown.



```

> vertex_attr(g6)
$name
[1] "47" "31" "12" "44" "25" "32" "3" "19" "33" "16" "43"
[12] "7" "27" "22" "34" "41" "35" "91" "37" "38" "39" "45"
[23] "40" "17" "15" "14" "21" "13" "5" "75" "106" "342" "72"
[34] "6" "24" "61" "8" "3543" "3552" "3548" "5797" "5799" "2942" "2941"
[45] "103"
> |

```

## 11. centr\_betw

This function centralizes a graph based on the betweenness of its vertices. As shown below, it returns the centrality scores of individual nodes, the centrality index of the overall graph and the theoretical maximum centrality index of the graph.

```

> centr_betw(g6, directed = TRUE)
$res
[1] 0.2142857 2.7609890 55.6221001 0.0000000 1.0000000 0.3482906
[7] 0.7054335 2.7054335 1.2054335 26.7142857 12.7142857 22.5476190
[13] 0.3482906 0.9554335 15.0943223 3.2142857 0.0000000 0.0000000
[19] 0.2142857 0.0000000 0.0000000 7.2142857 0.0000000 0.0000000
[25] 0.3333333 0.3333333 8.9316239 16.0000000 34.0982906 0.0000000
[31] 0.0000000 0.0000000 0.0000000 1.5705128 16.0000000 0.0000000
[37] 11.3333333 0.1538462 0.3333333 0.3333333 4.0000000 0.0000000
[43] 0.0000000 0.0000000 0.0000000

$centralization
[1] 0.02709968

$theoretical_max
[1] 83248

```

## Functions applied from the documentation:

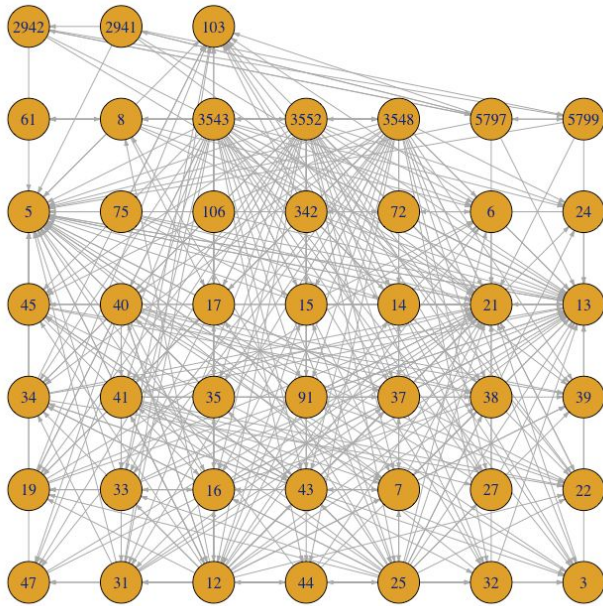
### 1. Simple grid layout:

This lets the user view every node in the graph in a simple grid layout. Dimensions can be specified to create a visualization as required.

```

> plot(g5, layout= layout_on_grid(g5, dim=2), edge.arrow.size=0.15)

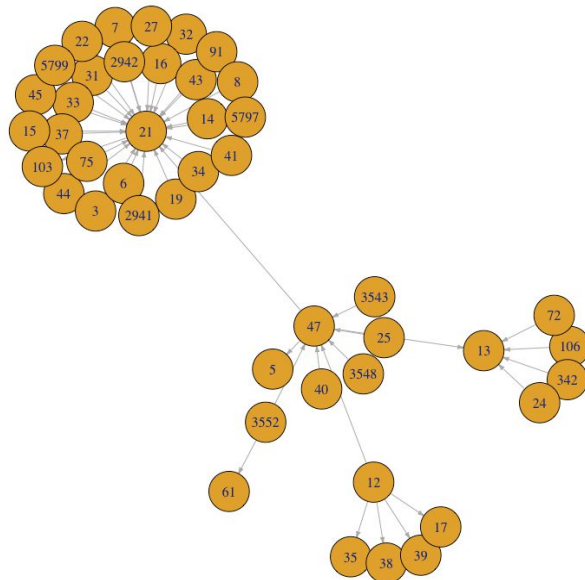
```



## 2. Minimum Spanning Tree

A walk is performed such that the subgraph contains vertices which are closest to each other.

```
> mstg5<-mst(g5, weights = NULL)
> plot(mstg5)
```





### 3. cluster\_walktrap

This function uses random walks to look for densely populated subgraphs(communities) in a graph.

```
> clu<-cluster_walktrap(g6, steps = 3)
> clu
IGRAPH clustering walktrap, groups: 7, mod: 0.082
+ groups:
  $`1`
  [1] "47" "31" "12" "44" "25" "32" "3" "19" "33" "16" "43"
  [12] "7" "27" "22" "34" "41" "35" "91" "37" "38" "39" "45"
  [23] "40" "15" "14" "6" "3543" "3552" "3548"

  $`2`
  [1] "17" "61" "8"

  $`3`
  [1] "21" "13" "5" "75" "103"
+ ... omitted several groups/vertices
> cluster_walktrap(g6, steps = 6)
IGRAPH clustering walktrap, groups: 7, mod: 0.082
+ groups:
  $`1`
  [1] "47" "31" "12" "44" "25" "32" "3" "19" "33" "16" "43"
  [12] "7" "27" "22" "34" "41" "35" "91" "37" "38" "39" "45"
  [23] "40" "15" "14" "6" "3543" "3552" "3548"

  $`2`
  [1] "21" "13" "5" "75" "103"

  $`3`
  [1] "17" "61" "8"
+ ... omitted several groups/vertices
> |
```

### 4. biconnected.components

This function finds the biconnected components of a graph and lists the number of those biconnected parts, the vertices of its components, its tree edges, component edges and articulation points.

```

> biconnected.components(g6)
$no
[1] 2

$tree_edges
$tree_edges[[1]]
+ 1/344 edge from 3647692 (vertex names):
[1] 106->13

$tree_edges[[2]]
+ 43/344 edges from 3647692 (vertex names):
[1] 2941->2942 2942->5799 5797->5799 5797->103 342 ->72 24 ->342
[7] 3548->24 75 ->14 75 ->15 3548->15 8 ->61 8 ->17
[13] 3548->17 45 ->39 45 ->38 3548->38 3548->35 3552->35
[19] 3552->27 3543->27 3543->91 91 ->22 40 ->22 40 ->37
[25] 43 ->37 7 ->43 16 ->7 33 ->16 33 ->19 41 ->19
[31] 41 ->32 25 ->32 25 ->44 12 ->44 12 ->3 34 ->3
[37] 31 ->34 31 ->103 6 ->103 5 ->6 13 ->5 21 ->13
[43] 47 ->21

$component_edges
$component_edges[[1]]
+ 1/344 edge from 3647692 (vertex names):
[1] 106->13

$component_edges[[2]]
+ 343/344 edges from 3647692 (vertex names):
[1] 2941->21 2941->13 2941->5 2941->5797 2941->5799 2941->2942
[7] 5797->2941 2942->21 2942->13 2942->5 2942->5797 2942->5799
[7] 5797->2941 2942->21 2942->13 2942->5 2942->5797 2942->5799
[13] 5797->2942 5799->21 5799->13 5799->5 5799->5797 5797->5799
[19] 5797->21 5797->13 5797->5 31 ->103 12 ->103 32 ->103
[25] 3 ->103 19 ->103 33 ->103 27 ->103 22 ->103 34 ->103
[31] 21 ->103 5 ->103 75 ->103 6 ->103 3543->103 5797->103
[37] 342 ->13 72 ->13 342 ->72 24 ->13 24 ->342 24 ->72
[43] 12 ->24 25 ->24 3548->47 3548->31 3548->12 3548->44
[49] 3548->25 3548->32 3548->3 3548->19 3548->33 3548->16
[55] 3548->43 3548->7 3548->27 3548->22 3548->34 3548->41
+ ... omitted several edges

$components
$components[[1]]
+ 2/45 vertices, named, from 3647692:
[1] 106 13

$components[[2]]
+ 44/45 vertices, named, from 3647692:
[1] 2941 2942 5799 5797 103 342 72 24 3548 75 14 15 8
[14] 61 17 45 39 38 35 3552 27 3543 91 22 40 37
[27] 43 7 16 33 19 41 32 25 44 12 3 34 31
[40] 6 5 13 21 47

$articulation_points
+ 1/45 vertex, named, from 3647692:
[1] 13
.

```

## 5. edge\_density

This calculates the ratio of the number of edges in a graph and the number of possible edges in a graph.

```
> edge_density(g6)
[1] 0.1737374
> diameter(g6)
[1] 4
> get_diameter(a6)
```

---

## 6. farthest\_vertices

This function returns the two vertices in the diameter path of a graph that are the farthest.

```
> farthest_vertices(g6)
$vertices
+ 2/45 vertices, named, from 15d2066:
[1] 33 45

$distance
[1] 4
```

---

## 7. dyad\_census

This measures and classifies the relationships between each different vertices in a graph into three categories- mutual, asymmetric and non-existent. These are the number of pairs with mutual connections, non-mutual connections, and no connections at all, respectively.

```
> clu<-cluster_walktrap(g6, steps = 3)
> clu
IGRAPH clustering walktrap, groups: 7, mod: 0.082
+ groups:
$`1`
[1] "47" "31" "12" "44" "25" "32" "3" "19" "33" "16" "43"
[12] "7" "27" "22" "34" "41" "35" "91" "37" "38" "39" "45"
[23] "40" "15" "14" "6" "3543" "3552" "3548"

$`2`
[1] "17" "61" "8"

$`3`
[1] "21" "13" "5" "75" "103"
+ ... omitted several groups/vertices
> cluster_walktrap(g6, steps = 6)
IGRAPH clustering walktrap, groups: 7, mod: 0.082
+ groups:
$`1`
[1] "47" "31" "12" "44" "25" "32" "3" "19" "33" "16" "43"
[12] "7" "27" "22" "34" "41" "35" "91" "37" "38" "39" "45"
[23] "40" "15" "14" "6" "3543" "3552" "3548"

$`2`
[1] "21" "13" "5" "75" "103"

$`3`
[1] "17" "61" "8"
+ ... omitted several groups/vertices
> |
```

---

## 8. incident\_edges

One can use this function to obtain a list of incoming and outgoing edges(given by the mode) incident on the specified vertex index.

```
> incident_edges(g5, 15, mode = "in")
$`34`
+ 9/344 edges from d15dbd9 (vertex names):
[1] 31 ->34 12 ->34 25 ->34 16 ->34 43 ->34 40 ->34 3543->34 3552->34 3548->34

> incident_edges(g5, 15, mode = "out")
$`34`
+ 6/344 edges from d15dbd9 (vertex names):
[1] 34->3 34->22 34->21 34->13 34->5 34->103

> incident_edges(g5, 15, mode = "all")
$`34`
+ 15/344 edges from d15dbd9 (vertex names):
[1] 34 ->3 34 ->22 34 ->21 34 ->13 34 ->5 34 ->103 31 ->34 12 ->34 25 ->34 16 ->34 43 ->34 40 ->34 3543->34
[14] 3552->34 3548->34
```

---

## 9. components

Calculates strongly connected components by performing two depth- first searches through the graph.

```
> components(g5, mode= "strong")
$membership
 47  31  12  44  25  32   3  19  33  16  43   7  27  22  34  41  35  91  37  38  39  45  40  17  15  14
 26  21   6  20   6  19  25  18  16  16  16  16  15  24  23   6  13   5  22  16  17  16   6  14  12  11
 21  13   5  75 106 342  72   6  24  61   8 3543 3552 3548 5797 5799 2942 2941 103
 27  27  27   4   3   9  10  27   8   7   7   2   2   2   1   1   1   1  28

$size
[1] 4 3 1 1 1 4 2 1 1 1 1 1 1 1 1 6 1 1 1 1 1 1 1 1 1 1 1 4 1

$no
[1] 28
```

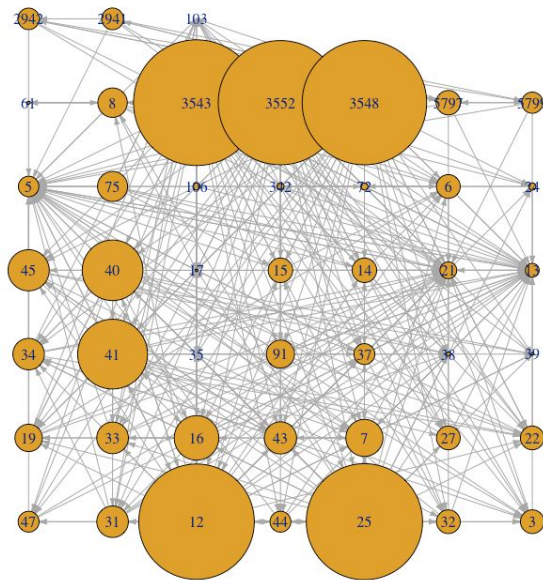
---

## 10. Hubs and Authorities

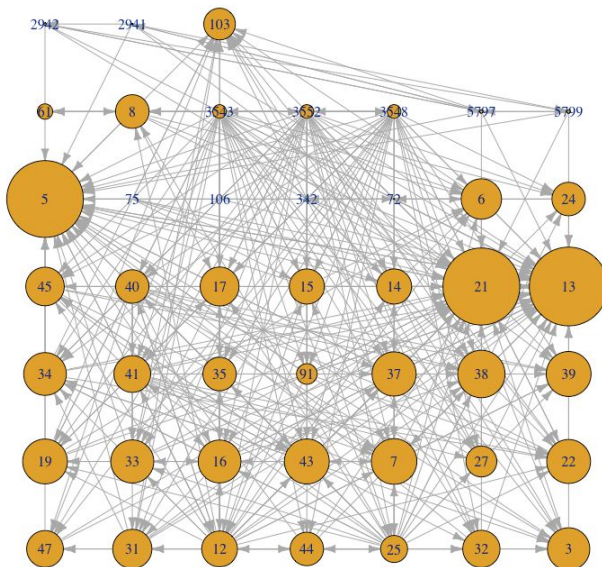
According to Kleinberg's measures of centrality, Authorities can be used to determine the nodes with important information, and Hubs point to these Authorities.

```
> hs <- hub_score(g5, weights=NA)$vector
> as <- authority_score(g5, weights=NA)$vector
> plot(g5, vertex.size=hs*50, main="Hubs", edge.arrow.size=0.15,layout= layout_on_grid(g5, dim=2))
> plot(g5, vertex.size=as*30, main="Authorities", edge.arrow.size=0.25,layout= layout_on_grid(g5, dim=2))
```

Hubs



Authorities



## 11. radius

The eccentricity of a vertex is its shortest path distance from the farthest other node in the graph. The smallest eccentricity in a graph is called its radius

```
> radius(g5)
[1] 2
```

the radius for our graph was 2

---

### 12. mean\_distance

calculates the average path length in a graph, by calculating the shortest paths between all pairs of vertices (both ways for directed graphs). This function does not consider edge weights currently and uses a breadth-first search.

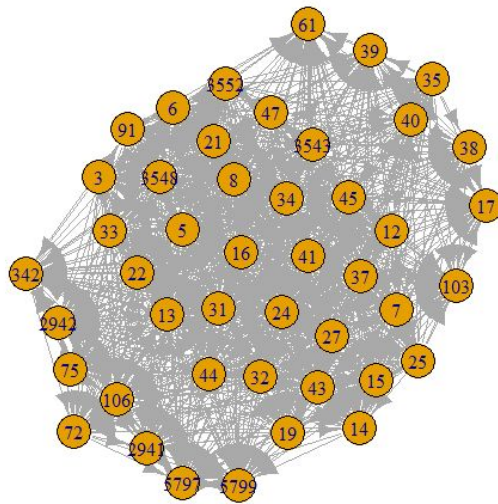
```
> mean_distance(g5)
[1] 1.453211
```

---

### 13. connect

creates a new graph by connecting each vertex to all other vertices in its neighborhood.

```
> connected_g5<-connect(g5, 2)
> plot(g5)
> |
```



it turn our graph into a highly connected structure

---

### 14. rewire

rewires the edges of a graph depending on the edge rewiring function

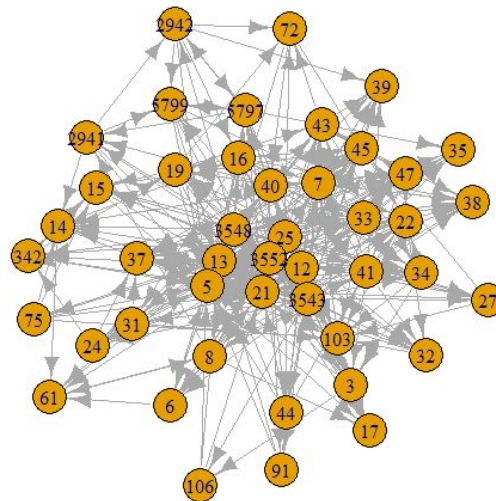


```

34 plot(rewire(g5, each_edge(p = .1, loops = FALSE)))
35

```

15



### 15.count\_triangles

counts how many triangles a vertex is part of

```

> V(g5)[40]
+ 1/45 vertex, named, from bf2139a:
[1] 3548
> count_triangles(g5,V(g5)[40])
[1] 247
> V(g5)[4]
+ 1/45 vertex, named, from bf2139a:
[1] 44
> count_triangles(g5,V(g5)[4])
[1] 28

```

given our original graph this make sense since node 44 close to the edge where as 3548 is near the center

## Question 6

### (a) central node(s) in the graph,

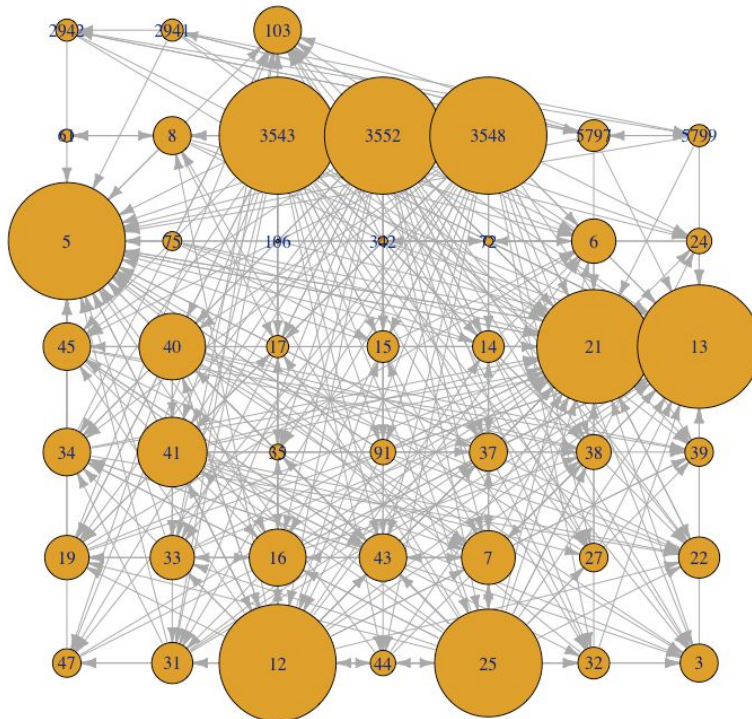
The centralization degree function centralizes a graph according to the degrees of vertices. The plot displays the nodes in sizes that vary with their measure of centrality.

```
> cent<-centr_degree(g5)
> plot(g5, vertex.size=cent$res, main="Centralities", edge.arrow.size=0.25,layout= layout_on_grid(g5, dim=2))
> cent
$res
[1] 9 13 37 8 34 10 12 14 14 18 15 17 9 13 15 22 5 8 12 11 9 15 21 7 10 10 36 39 37 6 1 3 3 14 8 4 12 37 37 10 7 7
[44] 7 15

$centralization
[1] 0.2755682

$theoretical_max
[1] 3872
```

Centralities



### (b) longest path

The longest path equals the diameter of the graph, and is of length 4. The list of vertices along the path is as below.

```

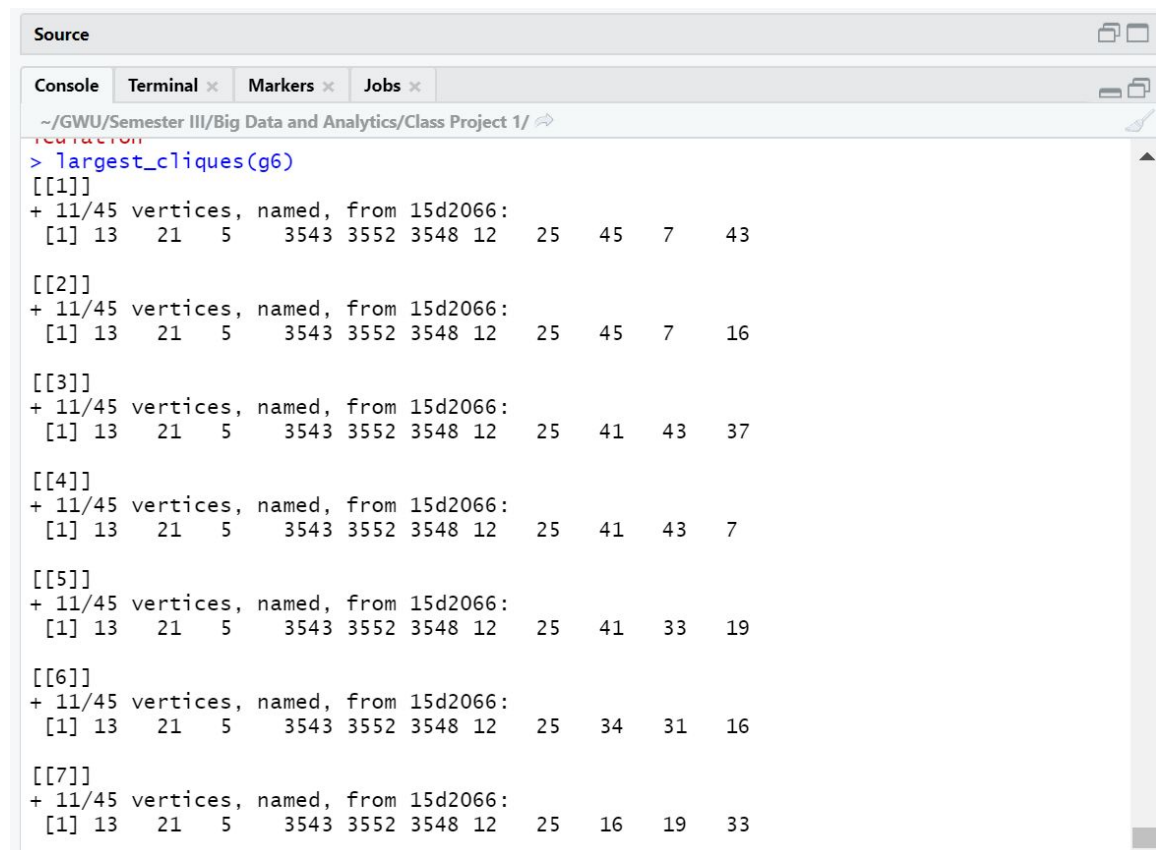
> diag5<-diameter(g5)
> diag5
[1] 4
> diag5Path<-get_diameter(g5)
> diag5Path
+ 5/45 vertices, named, from fe82803:
[1] 33 16 7 43 45

```

---

### (c) largest clique

There are 7 cliques which are the largest because they all consist of 11 vertices.



```

Source
Console Terminal × Markers × Jobs ×
~/GWU/Semester III/Big Data and Analytics/Class Project 1/
> largest_cliques(g6)
[[1]]
+ 11/45 vertices, named, from 15d2066:
[1] 13 21 5 3543 3552 3548 12 25 45 7 43

[[2]]
+ 11/45 vertices, named, from 15d2066:
[1] 13 21 5 3543 3552 3548 12 25 45 7 16

[[3]]
+ 11/45 vertices, named, from 15d2066:
[1] 13 21 5 3543 3552 3548 12 25 41 43 37

[[4]]
+ 11/45 vertices, named, from 15d2066:
[1] 13 21 5 3543 3552 3548 12 25 41 43 7

[[5]]
+ 11/45 vertices, named, from 15d2066:
[1] 13 21 5 3543 3552 3548 12 25 41 33 19

[[6]]
+ 11/45 vertices, named, from 15d2066:
[1] 13 21 5 3543 3552 3548 12 25 34 31 16

[[7]]
+ 11/45 vertices, named, from 15d2066:
[1] 13 21 5 3543 3552 3548 12 25 16 19 33

```

---

### (d) betweenness centrality

Calculates centrality based on shortest paths. Higher the centrality, higher the number of shortest paths passing through the vertex, as shown in the plot below.

```

> centr_betw(g5, directed = TRUE, nobigint = TRUE,
+           normalized = TRUE)
$res
[1] 0.2142857 2.7609890 55.6221001 0.0000000 1.0000000 0.3482906 0.7054335 2.7054335 1.2054335 26.7142857 12.7142857
[12] 22.5476190 0.3482906 0.9554335 15.0943223 3.2142857 0.0000000 0.0000000 0.2142857 0.0000000 0.0000000 7.2142857
[23] 0.0000000 0.0000000 0.3333333 0.3333333 8.9316239 16.0000000 34.0982906 0.0000000 0.0000000 0.0000000 0.0000000
[34] 1.5705128 16.0000000 0.0000000 11.3333333 0.1538462 0.3333333 0.3333333 4.0000000 0.0000000 0.0000000 0.0000000
[45] 0.0000000

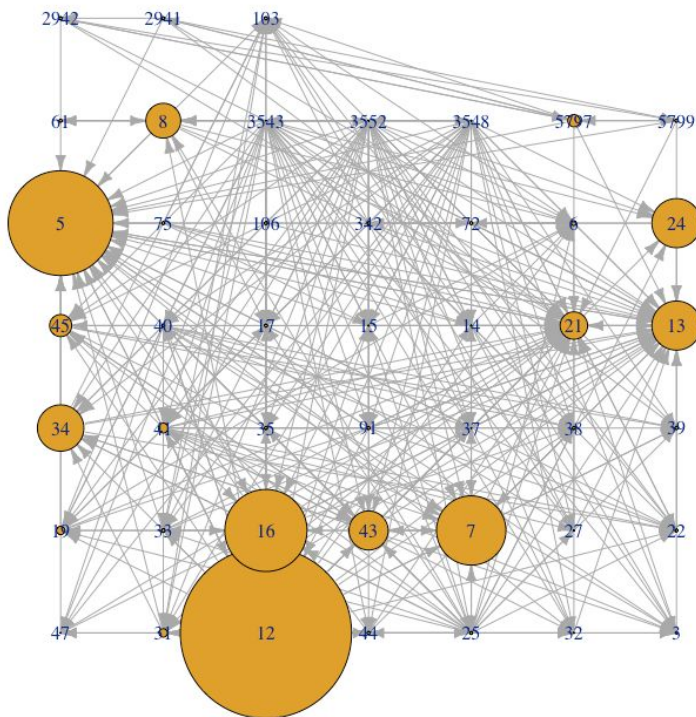
$centralization
[1] 0.02709968

$theoretical_max
[1] 83248

> bet<-centr_betw(g5, directed = TRUE, nobigint = TRUE,
+               normalized = TRUE)
> plot(g5, vertex.size=bet$res, main="Centralities", edge.arrow.size=0.25,layout= layout_on_grid(g5, dim=2))

```

### Betweenness Centrality



### (e) power centrality

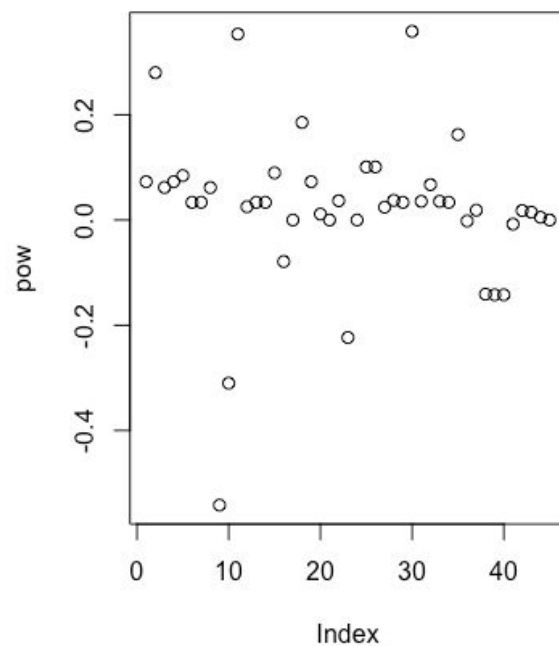
The power of a vertex is defined recursively by the sum of powers of its alters. Positive values indicate vertices become powerful as their vertices do so. Negative values imply a competitive relationship since vertices become powerful as their alters become weaker.

```

> pow<-power centrality(g5, nodes = V(g5), loops = FALSE,
+   exponent = 2, rescale = FALSE, tol = 1e-07, sparse = TRUE)
> pow
      47      31      12      44      25      32      3      19      33
-4.942921e-01 -1.901123e+00 -4.202127e-01 -4.942921e-01 -5.723026e-01 -2.281348e-01 -2.281348e-01 -4.182472e-01 3.677810e+00
      16      43      7      27      22      34      41      35      91
2.105062e+00 -2.398872e+00 -1.716772e-01 -2.281348e-01 -2.281348e-01 -6.083595e-01 5.352967e-01 0.000000e+00 -1.258673e+00
      37      38      39      45      40      17      15      14      21
-4.942921e-01 -7.719714e-02 0.000000e+00 -2.488743e-01 1.516019e+00 5.023397e-16 -6.844045e-01 -6.844045e-01 -1.647640e-01
      13      5      75      106      342      72      6      24      61
-2.534831e-01 -2.281348e-01 -2.433438e+00 -2.408090e-01 -4.562696e-01 -2.408090e-01 -2.281348e-01 -1.102652e+00 1.267416e-02
      8      3543      3552      3548      5797      5799      2942      2941      103
-1.267416e-01 9.565562e-01 9.650056e-01 9.650056e-01 5.293324e-02 -1.222683e-01 -1.006477e-01 -3.578585e-02 -1.891161e-15

```

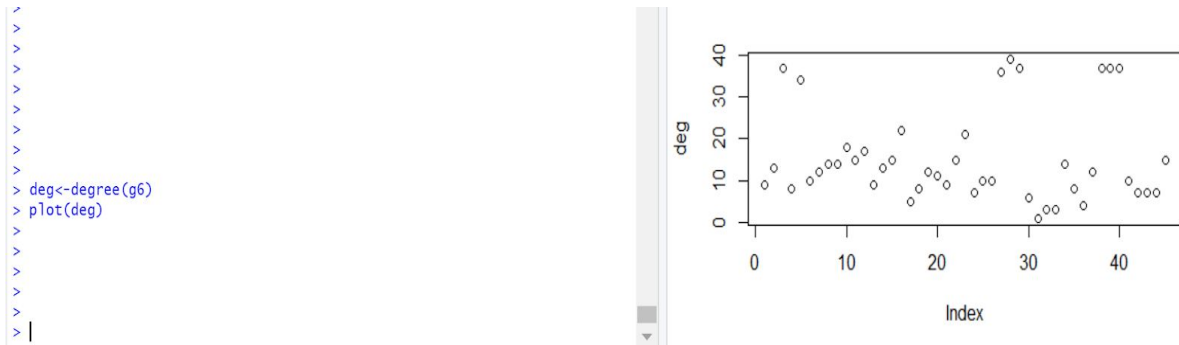
	V1
75	3.585261e-01
43	3.534334e-01
31	2.800985e-01
91	1.854442e-01
24	1.624571e-01
15	1.008355e-01
14	1.008355e-01
34	8.963153e-02
25	8.431914e-02
47	7.282562e-02
37	7.282562e-02
44	7.282562e-02



**a. Is there more than one node with the most degrees?**

There is only one node with the most degrees which is Node 13, which has 39 degrees.

```
> degree(g6)
 47   31   12   44   25   32    3   19   33   16   43    7   27   22
  9   13   37    8   34   10   12   14   14   18   15   17    9   13
 34   41   35   91   37   38   39   45   40   17   15   14   21   13
 15   22    5    8   12   11    9   15   21    7   10   10   36   39
  5   75  106  342  72    6   24   61    8 3543 3552 3548 5797 5799
 37    6    1    3    3   14    8    4   12   37   37   37   10    7
2942 2941  103
  7    7   15
>
> |
```



**b. Are there multiple longest paths?**

No, there is only one longest path which is the diameter of the graph.

```
> diameter(g6)
[1] 4
> get_diameter(g6)
+ 5/45 vertices, named, from 15d2066:
[1] 33 16 7 43 45
```

**c. Are there multiple cliques?**

There are multiple cliques present which means that many completely connected subgraphs exist in the graph. The screenshot below shows just a few of the many cliques that are of a minimum size of 10.



```

> cliques(g6, min = 10)
[[1]]
+ 10/45 vertices, named, from 15d2066:
[1] 12 25 16 34 21 13 5 3543 3552 3548

[[2]]
+ 10/45 vertices, named, from 15d2066:
[1] 12 25 16 45 21 13 5 3543 3552 3548

[[3]]
+ 10/45 vertices, named, from 15d2066:
[1] 12 25 41 37 21 13 5 3543 3552 3548

[[4]]
+ 10/45 vertices, named, from 15d2066:
[1] 12 25 33 16 21 13 5 3543 3552 3548

[[5]]
+ 10/45 vertices, named, from 15d2066:
[1] 12 25 33 41 21 13 5 3543 3552 3548

[[6]]
+ 10/45 vertices, named, from 15d2066:
[1] 12 25 33 21 13 5 6 8 3543 3552 3548

```

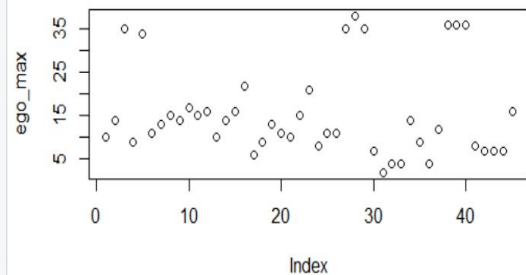
#### d. Are there one or multiple with the highest ego?

There is only one node with the highest ego of 38, as shown below.

```

>
>
>
>
>
>
> ego_max<-ego_size(g6)
> plot(ego_max)
>
>
>
>
>

```



#### e. What is the difference in betweenness centrality vs. power centrality for the cases you find? Consider comparing the nodes that are members of each set. Are there common nodes?

In each case what do you think the data tells you?

To compare, a matrix was created with the results of both centralities as columns.

```

> x<-matrix(centrbet$res)
> View(x)
> x<-cbind(x,pow)
> View(x)

```

The result was:

	V1	pow
40	0.0000000	-2.233599e-01
61	0.0000000	-1.867324e-03
17	0.0000000	-7.401130e-17
35	0.0000000	0.000000e+00
39	0.0000000	0.000000e+00
103	0.0000000	2.786308e-16
2941	0.0000000	5.272443e-03
38	0.0000000	1.137370e-02
2942	0.0000000	1.482875e-02
5799	0.0000000	1.801418e-02
106	0.0000000	3.547915e-02
72	0.0000000	3.547915e-02
342	0.0000000	6.722365e-02
44	0.0000000	7.282562e-02
91	0.0000000	1.854442e-01
75	0.0000000	3.585261e-01
3543	0.1538462	-1.409324e-01
37	0.2142857	7.282562e-02
47	0.2142857	7.282562e-02
3552	0.3333333	-1.421773e-01
3548	0.3333333	-1.421773e-01
15	0.3333333	1.008355e-01
14	0.3333333	1.008355e-01
32	0.3482906	3.361182e-02
27	0.3482906	3.361182e-02

3	0.7054335	3.361182e-02
22	0.9554335	3.361182e-02
25	1.0000000	8.431914e-02
33	1.2054335	-5.418633e-01
6	1.5705128	3.361182e-02
19	2.7054335	6.162168e-02
31	2.7609890	2.800985e-01
41	3.2142857	-7.886696e-02
5797	4.0000000	-7.798822e-03
45	7.2142857	3.666744e-02
21	8.9316239	2.427521e-02
8	11.3333333	1.867324e-02
43	12.7142857	3.534334e-01
34	15.0943223	8.963153e-02
13	16.0000000	3.734647e-02
24	16.0000000	1.624571e-01
7	22.5476190	2.529375e-02
16	26.7142857	-3.101455e-01
5	34.0982906	3.361182e-02
12	55.6221001	6.191126e-02

While betweenness centrality shows how often a node acts as a bridge on a shortest path between vertices, power centrality shows how strong or weak a node is based on its behaviour with respect to the growing and shrinking of other nodes over long walks. Node 103 has the lowest in both cases, implying it has no shortest paths passing through, and highly negative decay on growth of its alters.

Node 43 is one of the nodes that positively grows and also has a large betweenness centrality associated with it.

---

## Question 7

**Find the 20 nodes with the largest networks, e.g., having the greatest diameters. Do any of these circles overlap?**

how to get 20 node

```
ego<- make_ego_graph(g5, order = 1)
```

```
diameter_sub_graphs<-lapply(third_order, diameter)
```

```
as.matrix(diameter_sub_graphs)
```

this create subgraphs with each node with is direct nearest neighbors  
then each the diameter of each of those subgraphs

there are definitely overlaps

```
[[39]]
IGRAPH d3ab2bc DN-- 36 297 --
+ attr: name (v/c)
+ edges from d3ab2bc (vertex names):
[1] 47->21 47->13 47->5 31->34 31->37 31->21 31->13 31->5 12->47 12->31 12->44 12->25 12->32 12->3 12->19 12->33 12->16 12->43
[19] 12->7 12->22 12->34 12->41 12->35 12->37 12->38 12->39 12->45 12->40 12->17 12->15 12->14 12->21 12->13 12->5 12->6 12->24
[37] 12->8 44->21 44->13 44->5 25->47 25->31 25->12 25->44 25->32 25->3 25->19 25->33 25->16 25->43 25->7 25->27 25->22 25->34
[55] 25->41 25->35 25->37 25->38 25->39 25->45 25->40 25->17 25->15 25->14 25->21 25->13 25->5 25->6 25->24 25->8 32->21 32->13
[73] 32->5 3->21 3->13 3->5 19->21 19->13 19->5 19->6 33->19 33->16 33->21 33->13 33->5 16->31 16->19 16->33 16->7 16->34
[91] 16->38 16->21 16->13 16->5 43->34 43->37 43->45 43->21 43->13 43->5 7->16 7->43 7->38 7->39 7->21 7->13 7->5 27->21
[109] 27->13 27->5 22->21 22->13 22->5 34->3 34->22 34->21 34->13 34->5 41->12 41->32 41->3 41->19 41->33 41->43 41->7 41->27
[127] 41->22 41->37 41->38 41->39 41->17 41->21 41->13 41->5 91->12 91->22 91->21 91->13 91->5 37->21 37->13 37->5 38->7 45->16
+ ... omitted several edges

[[40]]
IGRAPH d3ab2bc DN-- 36 297 --
+ attr: name (v/c)
+ edges from d3ab2bc (vertex names):
[1] 47->21 47->13 47->5 31->34 31->37 31->21 31->13 31->5 12->47 12->31 12->44 12->25 12->32 12->3 12->19 12->33 12->16 12->43
[19] 12->7 12->22 12->34 12->41 12->35 12->37 12->38 12->39 12->45 12->40 12->17 12->15 12->14 12->21 12->13 12->5 12->6 12->24
[37] 12->8 44->21 44->13 44->5 25->47 25->31 25->12 25->44 25->32 25->3 25->19 25->33 25->16 25->43 25->7 25->27 25->22 25->34
[55] 25->41 25->35 25->37 25->38 25->39 25->45 25->40 25->17 25->15 25->14 25->21 25->13 25->5 25->6 25->24 25->8 32->21 32->13
[73] 32->5 3->21 3->13 3->5 19->21 19->13 19->5 19->6 33->19 33->16 33->21 33->13 33->5 16->31 16->19 16->33 16->7 16->34
[91] 16->38 16->21 16->13 16->5 43->34 43->37 43->45 43->21 43->13 43->5 7->16 7->43 7->38 7->39 7->21 7->13 7->5 27->21
[109] 27->13 27->5 22->21 22->13 22->5 34->3 34->22 34->21 34->13 34->5 41->12 41->32 41->3 41->19 41->33 41->43 41->7 41->27
[127] 41->22 41->37 41->38 41->39 41->17 41->21 41->13 41->5 91->12 91->22 91->21 91->13 91->5 37->21 37->13 37->5 38->7 45->16
+ ... omitted several edges
```

---

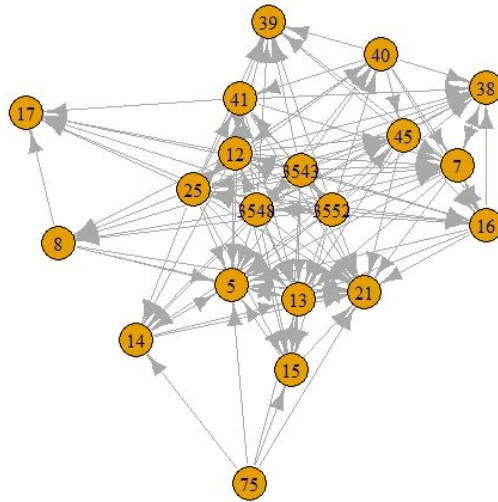
**a. Build a matrix of 20 nodes with their reachability to the 3<sup>rd</sup> level**

```
g6<-delete_vertices(g5,
```

```
V(g5)[4,17,18,31,32,33,34,35,36,41,42,43,44,45,1,2,6,7,8,9,11,13,14,15,19])
```

plot(g6)

---



---

**b. Determine which of the 20 nodes share common nodes, if any, and, for each common node, list the nodes that share that common node.**

To see which nodes are in common I created an Adjacency matrix from the 3rd order graphs.

```

> as_adjacency_matrix(g7)
20 x 20 sparse Matrix of class "dgCMatrix"
  [[ suppressing 20 column names '12', '25', '16'

12  . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . 1 . . .
25  1 . 1 1 1 1 1 1 1 1 1 1 1 1 1 . 1 . . .
16  . . . 1 . 1 . . . . . 1 1 1 . . . . .
7   . . 1 . . 1 1 . . . . . 1 1 1 . . . . .
41  1 . . 1 . 1 1 . . 1 . . 1 1 1 . . . . .
38  . . . 1 . . . . . . . . . . . . . . .
39  . . . . . . . . . . . . . . . . . . .
45  . . 1 1 . 1 1 . . . . . 1 1 1 . . . . .
40  1 . 1 1 1 1 1 1 1 . . . . . . . . . . .
17  . . . . . . . . . . . . . . . . . . .
15  . . . . . . . . . . . 1 1 1 . . . . .
14  . . . . . . . . . . . 1 1 1 . . . . .
21  . . . . . . . . . . . 1 1 . . . . .
13  . . . . . . . . . . . 1 . 1 . . . . .
5   . . . . . . . . . . . 1 1 . . . . .
75  . . . . . . . . . 1 1 1 1 1 . . . . .
8   . . . . . . . . . 1 . . 1 1 1 . . . . .
3543 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . 1 . 1 1
3552 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . 1 1 . 1
3548 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . 1 1 1 .
> |

```

if you read the matrix going down you can see looking at the matrix you can see that nodes 12 and 25 have 16 in common because there is a 1 in the 3rd column. There are too many shared nodes to list out so i'm using the matrix as my list.

---