Graphs 1: Graphs and their parameters

300958 Social Web Analysis

Week 6 Lab Solutions

- By visually examining the two graphs above, which looks denser? Use the function `graph.density` to compute the density of each graph and compare the results to your guess.

The ER graph looks to have higher density since it contains many more edges.

```
library("igraph", quietly=TRUE)
```

```
##
## Attaching package: 'igraph'
##
## The following objects are masked from 'package:stats':
##
##     decompose, spectrum
##
## The following object is masked from 'package:base':
##
##     union
```



```
g.er = erdos.renyi.game(n = 100, p = 0.1)
g.ba = barabasi.game(n = 100, directed=FALSE)
graph.density(g.er)
```

```
## [1] 0.1030303
```

```
graph.density(g.ba)
```

```
## [1] 0.02
```

- The diameter is the longest shortest path. Which of the two graphs do you expect to have the largest diameter? Use the function `diameter` to compute the diameter of each graph.

We expect the ER graph to have a smaller diameter because there are many paths between each of the vertices.

```
diameter(g.er)
```

```
## [1] 4
```

```
diameter(g.ba)
```

```
## [1] 13
```

- What do you expect the degree distribution of each graph to look like? We can compute the degree of each vertex using the function `degree`. We can also compute the degree distribution of the graph using the function `degree.distribution`. Use the `help` pages to understand the output.

The ER degree distribution should look mound shaped (a mean with left and right tails). The BA degree distribution should look exponentially decaying (many vertices with low degree, a few with high degree).

```
degree(g.er)
```

```
##  [1] 13 11  8  8  6 17 11 12 10 16 10 14  7 11 10  6  8 11  6 13  6  7 10
## [24] 12 15  9 10 14  9 11  8  7  6 10 12  9  6 12 13  4  7 13 16 12 10  9
## [47] 14 10 14 16  8 10 10 10  8  9 10  8 14  7 14  8  9 12 13  8  8 13 11
## [70] 13  7  8 11 10  4 10 10  6  9 10  7  8 13 13 15  7 11 12  8 12  5 12
## [93] 12  9 11 14 10 10 12 12
```



```
degree.distribution(g.er)
```

```
## [1] 0.00 0.00 0.00 0.00 0.02 0.01 0.07 0.08 0.13 0.08 0.18 0.09 0.12 0.09
## [15] 0.07 0.02 0.03 0.01
```

```
degree(g.ba)
```

```
##  [1]  8  5  4  4  4  3 11  2  2  4  8  1  4  6  2  3  4  3  1  1  2  1  1
## [24]  2  1  6  1  1  5  1  2  2  5  3  1  2  1  1  1  1  3  1  1  3  2  1
## [47]  1  1  3  1  1  2  2  1  1  1  2  1  1  2  2  1  1  3  1  1  1  1  2
## [70]  1  1  1  1  1  1  1  1  1  2  1  1  2  1  1  1  1  2  1  1  1  1  1
## [93]  1  1  1  1  1  1  1  1
```

```
degree.distribution(g.ba)
```

```
## [1] 0.00 0.60 0.18 0.08 0.06 0.03 0.02 0.00 0.02 0.00 0.00 0.01
```

- Which vertex is most central according to Degree Centrality?

To find the most central, we order the vertices by their degree.

```
order(degree(g.er), decreasing=TRUE)
```

```
##   [1]   6  10  43  50  25  85  12  28  47  49  59  61  96   1  20  39  42
##  [18]  65  68  70  83  84   8  24  35  38  44  64  88  90  92  93  99 100
##  [35]   2   7  14  18  30  69  73  87  95   9  11  15  23  27  34  45  48
##  [52]  52  53  54  57  74  76  77  80  97  98  26  29  36  46  56  63  79
##  [69]  94   3   4  17  31  51  55  58  62  66  67  72  82  89  13  22  32
##  [86]  41  60  71  81  86   5  16  19  21  33  37  78  91  40  75
```

```
order(degree(g.ba), decreasing=TRUE)
```

```
##   [1]   7   1  11  14  26   2  29  33   3   4   5  10  13  17   6  16  18
##  [18]  34  41  44  49  64   8   9  15  21  24  31  32  36  45  52  53  57
##  [35]  60  61  69  79  82  87  12  19  20  22  23  25  27  28  30  35  37
##  [52]  38  39  40  42  43  46  47  48  50  51  54  55  56  58  59  62  63
##  [69]  65  66  67  68  70  71  72  73  74  75  76  77  78  80  81  83  84
##  [86]  85  86  88  89  90  91  92  93  94  95  96  97  98  99 100
```

- Read the R help page for `closeness` to find what R is computing. Then work out which of the vertices is the most central with respect to closeness centrality.

The R function closeness provides the reciprocal of the sum of path lengths. Therefore the sum of path lengths is:


```
1/closeness(g.ba)
```



```
##   [1] 347 411 401 355 437 441 463 447 483 447 381 561 555 557 553 649 645
##  [18] 475 545 653 533 479 445 443 747 453 539 551 537 561 631 745 463 469
##  [35] 499 559 653 561 541 743 557 551 635 651 559 655 657 655 541 843 539
##  [52] 549 633 535 561 657 497 479 595 477 727 639 731 563 631 743 561 635
##  [69] 559 561 743 551 661 545 575 573 545 509 651 445 639 747 661 825 749
##  [86] 561 565 845 749 663 653 535 561 509 655 647 573 657 655 509
```

We want the vertex with the shortest path lengths, therefore we want the maximum given by the R closeness function.

```
order(closeness(g.er), decreasing = TRUE)
```

```
##   [1]  50  43   6  28  10  25  85  12  49  61  42  47  84  59  65  99  68
##  [18]  83  20  39  64  35 100   1  24  44  69   8  87  88  92  30  36  93
##  [35]  74  90  96  18   2   7  11  48  57  70  73  27  76  38  77  94  23
##  [52]  80  14  15  72  95  97   9  26  56  45  46  54   4  34  53   3  98
##  [69]  55  52  79  29  82  31  62  67  17  51  58  63  66  60  32  41  89
##  [86]  19  86  81  78  16  37  71  13  22   5  33  91  21  40  75
```

```
order(closeness(g.ba), decreasing = TRUE)
```

```
##   [1]   1   4  11   3   2   5   6  24  23  80   8  10  26   7  33  34  18
##  [18]  60  22  58   9  57  35  78  94 100  21  54  92  29  27  51  39  49
##  [35]  19  74  77  52  28  42  72  15  13  14  41  36  45  69  12  30  38
##  [52]  55  67  70  86  93  64  87  76  97  75  59  31  65  53  43  68  62
##  [69]  81  17  96  16  44  79  20  37  91  46  48  95  99  47  56  98  73
##  [86]  83  90  61  63  40  66  71  32  25  82  85  89  84  50  88
```

- Is the centre the same for all three centrality measures? Examine this for the Erdős-Renyi graph and Barabási–Albert graph.

```
order(betweenness(g.er), decreasing = TRUE)
```

```
##   [1]  10  43  85   6  61  96  50  49  59  84  88  28  47  20  25  39  24
##  [18]  83  99  64  12  42  65   2  70  93   1  68  35   8  87  92   7  52
##  [35]  95 100  69  44  27  90  73  14  38  45  97  23   9  36  18  74  34
##  [52]  11  98  15  30  53  62  48  77  76  79  57  80  94  56  89  17  29
##  [69]  26  66  60  54  63  31  46  22  81  58  67  82   3  13  71  37  72
##  [86]  86  41  78  40  51  55  21  33  10   5  16  32  91  90  75
```



```
order(betweenness(g.ba), decreasing = TRUE)
```

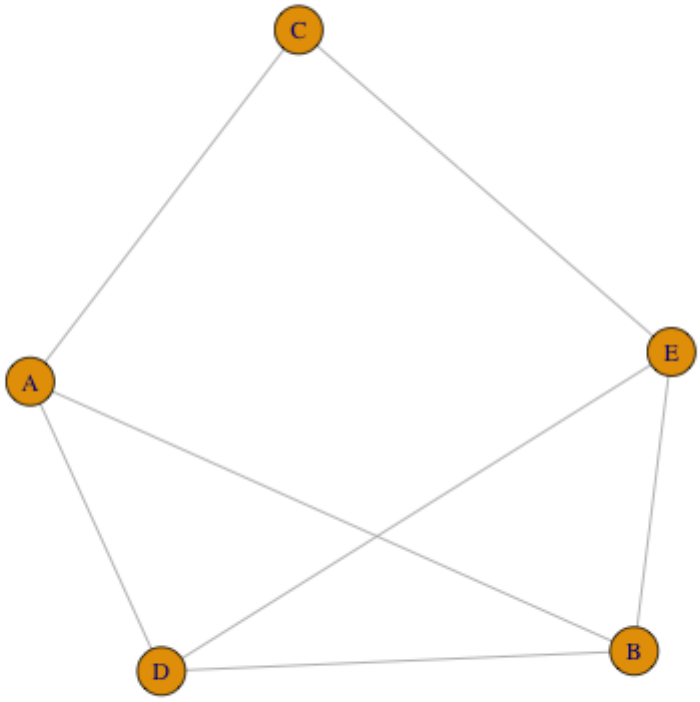


```
##   [1]   1  11   4   3   7   2  26   9  14  33  29  34   5  15  10  13  17
##  [18]  16   8   6  18  41  44  49  64   8  70  21  24  28  36  45  52  53
##  [35]  57  60  61  69  82  87  12  19  20  22  23  25  27  28  30  35  37
##  [52]  38  39  40  42  43  46  47  48  50  51  54  55  56  58  59  62  63
##  [69]  65  66  67  68  70  71  72  73  74  75  76  77  78  80  81  83  84
##  [86]  85  86  88  89  90  91  92  93  94  95  96  97  98  99 100
```


Compare the above orders.

- Using the following graph:

```
g3 = graph.formula(A-B, A-C, A-D, B-D, B-E, E-D, C-E)
plot(g3)
```



Small Graph

Calculate the Degree Distribution, Degree Centrality, Closeness Centrality, Betweenness Centrality using the methods shown in the lecture. Then check your answer using the R functions.

```
degree.distribution(g3)
```

```
## [1] 0.0 0.0 0.2 0.8
```

```
degree(g3)
```

```
## A B C D E
## 3 3 2 3 3
```

```
1/closeness(g3)
```

```
## A B C D E
## 5 5 6 5 5
```

```
betweenness(g3)
```

```
##         A         B         C         D         E
## 1.0000000 0.3333333 0.3333333 0.3333333 1.0000000
```

- Find the 10 friends that have the most followers. What are their names? Note the function `sort` will sort the vector of follower counts. The function `order` will sort, but provide the position of the sort. So to find the top 10, we use `order` with `decreasing=TRUE` and choose the first ten values, giving us the positions of the top 10.

```
friendFollowCount = count.followers(friends)
friendPosition = order(friendFollowCount, decreasing = TRUE)[1:10]
topFriends = friends[friendPosition]
```

- Write a `for` loop to check if any of the friend accounts are protected and store the `TRUE/FALSE` values in the variable `protected.status`.

```
n = length(friends)
protected.status = rep(0, n)
for (a in 1:n) {
  protected.status[a] = friends[[a]]$getProtected()
}
```


- Using the function `count.followers` find the 10 unprotected friends with the most followers and store them in `top.unprotected.friends`.

```
friendFollowCount = count.followers(unprotected.friends)
friendPosition = order(friendFollowCount, decreasing = TRUE)[1:10]
top.unprotected.friends = unprotected.friends[friendPosition]
```

- Write a `for` loop to download 100 friends from the 10 most popular friends of Wil Wheaton and store them in `more.friends`.

```
n = length(top.unprotected.friends)
for (a in 1:n) {
  more.friends[[a]] = top.unprotected.friends[[a]]$getFriends(100)
}
```

- Write a `for` loop to store all 100 screen names in the variable `friend.names`.

```
friend.names = c()
n = length(friends)
for (a in 1:n) {
  friend.names[a] = friends[[a]]$getScreenName()
}
```

- Using what you have done above, write the function:

```
user.to.edgelist <- function(user, friends) {

  # create the list of friend screen names
  friend.names = c()
  for (a in c(1:length(friends))) {
    friend.names[a] = friends[[a]]$getScreenName()
  }

  user.name = rep(user$getScreenName(), length(friends)) # repeat user's name
  el = cbind(user.name,friend.names) # bind the columns to create a matrix

  return(el)
}
```

- Who is at the centre of the graph? Use the centrality measures to examine this.

```
order(closeness(g2), decreasing=TRUE)
```

- Examine the graph density. Is it sparse or dense?


```
density(g2)
```


- Examine the degree distribution. Is this graph more similar to an Erdős-Renyi graph or a Barabási–Albert graph?

```
degree.distribution(g2)
```