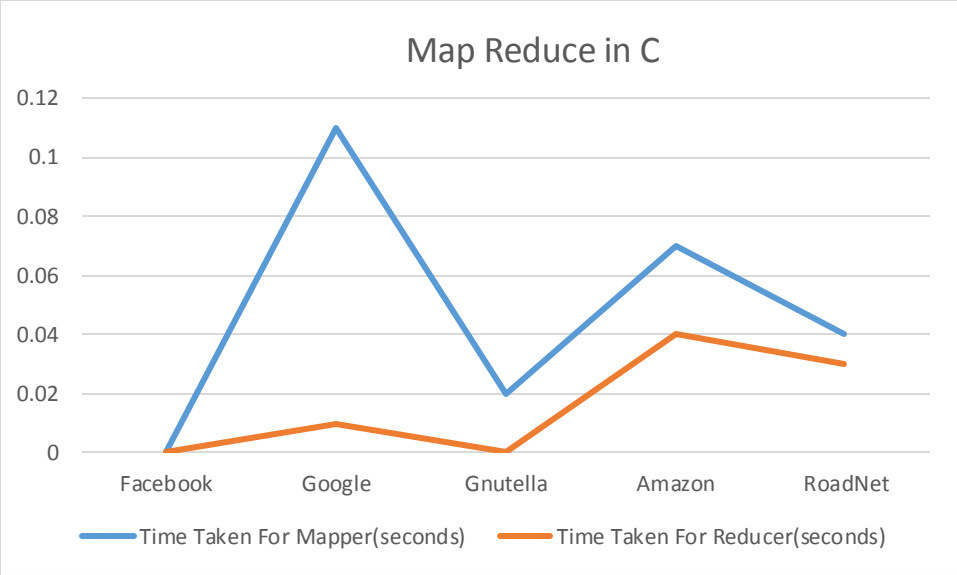
Problem 1: The goal of this assignment is to make sure you know how to use debugger and profiler tools. You are asked to write a C/C++ code for calculating the degree of each vertex in the graph G with *n* vertices and *m* edges. It will be a serial program but it must mimick the map/reduce API. The graphs will be provided to you for testing. The graphs will be stored in ASCII files: one file per graph. The file format is the following: (a) the first line stores *n* and *m*, the number of vertices and edges, resp.; (b) every other line is for an edge defined by its vertex id pair; (c) each vertex id is an integer between 0 and *n*-1.

(1) ***Demonstrate that you know how to use the profiler (10 pts):*** Profile the performance of your program using the GNU ***gprof*** profiler (https://sourceware.org/binutils/docs/gprof/):

(a) Using small, medium, large size graphs provided to you, write a short summary from the profiler. Specifically comment on the cost of performing the I/O versus computation. Which functions take most of the computational time? Are there any functions that take longer (e.g., say 80% of the total time)?

Graph Type	Time Taken For Mapper(seconds)	Time Taken For Reducer(seconds)
Facebook	0	0
Google	0.11	0.01
Gnutella	0.02	0
Amazon	0.07	0.04
RoadNet	0.04	0.03



As we see google graph with highest number of edges and vertices takes highest amount of time to compute the degree of each node. As we see as the size of the graph increases the amount of time taken to compute the degree also increases. Facebook Graph takes the least amount of time to run.

I/O Operations: Since mapper function involves two heavy IO operation for storing and reading from the file (initializeMapperDataStructure() and storeIntermediateResults()) mapper function takes more time that reducer. Mapper function also does a lot of housekeeping operations such as initializing data structures etc. Hence it takes more time that reducer functionality. It is seen that CPU operations generally do not take as much time as IO operations from the results of profiler as most of the time is spent in IO.

For amazon Graph,

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 36.48      0.04     0.04        1    40.13    40.13  ReduceFunction
 36.48      0.08     0.04        1    40.13    40.13  initializeMapperDataStructure
 18.24      0.10     0.02        1    20.06    20.06  storeIntermediateResults
  9.12      0.11     0.01   925872     0.00     0.00  simulateMapperFunction
  0.00      0.11     0.00        1     0.00    70.22  MapperFunction
  0.00      0.11     0.00        1     0.00     0.00  initializeDegreeArray
  0.00      0.11     0.00        1     0.00     0.00  initializeVerticesAndEdges
  0.00      0.11     0.00        1     0.00     0.00  outputResults
```

```
granularity: each sample hit covers 2 byte(s) for 9.06% of 0.11 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00    0.11                 main [1]
                0.00    0.07       1/1           MapperFunction [2]
                0.04    0.00       1/1           ReduceFunction [3]
-----------------------------------------------
                0.00    0.07       1/1           main [1]
[2]     63.6    0.00    0.07       1         MapperFunction [2]
                0.04    0.00       1/1           initializeMapperDataStructure [4]
                0.02    0.00       1/1           storeIntermediateResults [5]
                0.01    0.00  925872/925872      simulateMapperFunction [6]
                0.00    0.00       1/1           initializeVerticesAndEdges [8]
-----------------------------------------------
                0.04    0.00       1/1           main [1]
[3]     36.4    0.04    0.00       1         ReduceFunction [3]
                0.00    0.00       1/1           initializeDegreeArray [7]
                0.00    0.00       1/1           outputResults [9]
-----------------------------------------------
                0.04    0.00       1/1           MapperFunction [2]
[4]     36.4    0.04    0.00       1         initializeMapperDataStructure [4]
-----------------------------------------------
                0.02    0.00       1/1           MapperFunction [2]
[5]     18.2    0.02    0.00       1         storeIntermediateResults [5]
-----------------------------------------------
                0.01    0.00  925872/925872      MapperFunction [2]
[6]      9.1    0.01    0.00  925872          simulateMapperFunction [6]
-----------------------------------------------
                0.00    0.00       1/1           ReduceFunction [3]
[7]      0.0    0.00    0.00       1         initializeDegreeArray [7]
-----------------------------------------------
                0.00    0.00       1/1           MapperFunction [2]
[8]      0.0    0.00    0.00       1         initializeVerticesAndEdges [8]
-----------------------------------------------
                0.00    0.00       1/1           ReduceFunction [3]
[9]      0.0    0.00    0.00       1         outputResults [9]
-----------------------------------------------
```

For Facebook Graph,

```
Flat profile:

Each sample counts as 0.01 seconds.
 no time accumulated

  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
  0.00      0.00     0.00    84243     0.00     0.00  simulateMapperFunction
  0.00      0.00     0.00        1     0.00     0.00  MapperFunction
  0.00      0.00     0.00        1     0.00     0.00  ReduceFunction
  0.00      0.00     0.00        1     0.00     0.00  initializeDegreeArray
  0.00      0.00     0.00        1     0.00     0.00  initializeMapperDataStructure
  0.00      0.00     0.00        1     0.00     0.00  initializeVerticesAndEdges
  0.00      0.00     0.00        1     0.00     0.00  outputResults
  0.00      0.00     0.00        1     0.00     0.00  storeIntermediateResults
```

```
granularity: each sample hit covers 2 byte(s) no time propagated

index % time    self  children    called     name
                0.00    0.00   84243/84243       MapperFunction [2]
[1]      0.0    0.00    0.00   84243         simulateMapperFunction [1]
-----------------------------------------------
                0.00    0.00       1/1           main [14]
[2]      0.0    0.00    0.00       1         MapperFunction [2]
                0.00    0.00   84243/84243       simulateMapperFunction [1]
                0.00    0.00       1/1           initializeMapperDataStructure [5]
                0.00    0.00       1/1           initializeVerticesAndEdges [6]
                0.00    0.00       1/1           storeIntermediateResults [8]
-----------------------------------------------
                0.00    0.00       1/1           main [14]
[3]      0.0    0.00    0.00       1         ReduceFunction [3]
                0.00    0.00       1/1           initializeDegreeArray [4]
                0.00    0.00       1/1           outputResults [7]
-----------------------------------------------
                0.00    0.00       1/1           ReduceFunction [3]
[4]      0.0    0.00    0.00       1         initializeDegreeArray [4]
-----------------------------------------------
                0.00    0.00       1/1           MapperFunction [2]
[5]      0.0    0.00    0.00       1         initializeMapperDataStructure [5]
-----------------------------------------------
                0.00    0.00       1/1           MapperFunction [2]
[6]      0.0    0.00    0.00       1         initializeVerticesAndEdges [6]
-----------------------------------------------
                0.00    0.00       1/1           ReduceFunction [3]
[7]      0.0    0.00    0.00       1         outputResults [7]
-----------------------------------------------
                0.00    0.00       1/1           MapperFunction [2]
[8]      0.0    0.00    0.00       1         storeIntermediateResults [8]
-----------------------------------------------
```

For Google Graph,

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 40.13      0.10     0.10        1   100.31   110.34  ReduceFunction
 18.06      0.15     0.05  4322051     0.00     0.00  simulateMapperFunction
 18.06      0.19     0.05        1    45.14    45.14  initializeMapperDataStructure
 12.04      0.22     0.03        1    30.09   140.44  MapperFunction
  8.03      0.24     0.02        1    20.06    20.06  storeIntermediateResults
  4.01      0.25     0.01        1    10.03    10.03  outputResults
  0.00      0.25     0.00        1     0.00     0.00  initializeDegreeArray
  0.00      0.25     0.00        1     0.00     0.00  initializeVerticesAndEdges
```

```
granularity: each sample hit covers 2 byte(s) for 3.99% of 0.25 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00    0.25                 main [1]
                0.03    0.11       1/1           MapperFunction [2]
                0.10    0.01       1/1           ReduceFunction [3]
-----------------------------------------------
                0.03    0.11       1/1           main [1]
[2]     56.0    0.03    0.11       1         MapperFunction [2]
                0.05    0.00 4322051/4322051     simulateMapperFunction [4]
                0.05    0.00       1/1           initializeMapperDataStructure [5]
                0.02    0.00       1/1           storeIntermediateResults [6]
                0.00    0.00       1/1           initializeVerticesAndEdges [9]
-----------------------------------------------
                0.10    0.01       1/1           main [1]
[3]     44.0    0.10    0.01       1         ReduceFunction [3]
                0.01    0.00       1/1           outputResults [7]
                0.00    0.00       1/1           initializeDegreeArray [8]
-----------------------------------------------
                0.05    0.00 4322051/4322051     MapperFunction [2]
[4]     18.0    0.05    0.00 4322051         simulateMapperFunction [4]
-----------------------------------------------
                0.05    0.00       1/1           MapperFunction [2]
[5]     18.0    0.05    0.00       1         initializeMapperDataStructure [5]
-----------------------------------------------
                0.02    0.00       1/1           MapperFunction [2]
[6]      8.0    0.02    0.00       1         storeIntermediateResults [6]
-----------------------------------------------
                0.01    0.00       1/1           ReduceFunction [3]
[7]      4.0    0.01    0.00       1         outputResults [7]
-----------------------------------------------
                0.00    0.00       1/1           ReduceFunction [3]
[8]      0.0    0.00    0.00       1         initializeDegreeArray [8]
-----------------------------------------------
                0.00    0.00       1/1           MapperFunction [2]
[9]      0.0    0.00    0.00       1         initializeVerticesAndEdges [9]
-----------------------------------------------
```

For Gnutella Graph,

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
100.31      0.02     0.02        1    20.06    20.06  storeIntermediateResults
  0.00      0.02     0.00   147892     0.00     0.00  simulateMapperFunction
  0.00      0.02     0.00        1     0.00    20.06  MapperFunction
  0.00      0.02     0.00        1     0.00     0.00  ReduceFunction
  0.00      0.02     0.00        1     0.00     0.00  initializeDegreeArray
  0.00      0.02     0.00        1     0.00     0.00  initializeMapperDataStructure
  0.00      0.02     0.00        1     0.00     0.00  initializeVerticesAndEdges
  0.00      0.02     0.00        1     0.00     0.00  outputResults
```

```
granularity: each sample hit covers 2 byte(s) for 49.84% of 0.02 seconds

index % time    self  children    called     name
                0.00    0.02       1/1           main [3]
[1]    100.0    0.00    0.02       1         MapperFunction [1]
                0.02    0.00       1/1           storeIntermediateResults [2]
                0.00    0.00  147892/147892      simulateMapperFunction [4]
                0.00    0.00       1/1           initializeMapperDataStructure [7]
                0.00    0.00       1/1           initializeVerticesAndEdges [8]
-----------------------------------------------
                0.02    0.00       1/1           MapperFunction [1]
[2]    100.0    0.02    0.00       1         storeIntermediateResults [2]
-----------------------------------------------
                                                 <spontaneous>
[3]    100.0    0.00    0.02                 main [3]
                0.00    0.02       1/1           MapperFunction [1]
                0.00    0.00       1/1           ReduceFunction [5]
-----------------------------------------------
                0.00    0.00  147892/147892      MapperFunction [1]
[4]      0.0    0.00    0.00  147892         simulateMapperFunction [4]
-----------------------------------------------
                0.00    0.00       1/1           main [3]
[5]      0.0    0.00    0.00       1         ReduceFunction [5]
                0.00    0.00       1/1           initializeDegreeArray [6]
                0.00    0.00       1/1           outputResults [9]
-----------------------------------------------
                0.00    0.00       1/1           ReduceFunction [5]
[6]      0.0    0.00    0.00       1         initializeDegreeArray [6]
-----------------------------------------------
                0.00    0.00       1/1           MapperFunction [1]
[7]      0.0    0.00    0.00       1         initializeMapperDataStructure [7]
-----------------------------------------------
                0.00    0.00       1/1           MapperFunction [1]
[8]      0.0    0.00    0.00       1         initializeVerticesAndEdges [8]
-----------------------------------------------
                0.00    0.00       1/1           ReduceFunction [5]
[9]      0.0    0.00    0.00       1         outputResults [9]
-----------------------------------------------
```

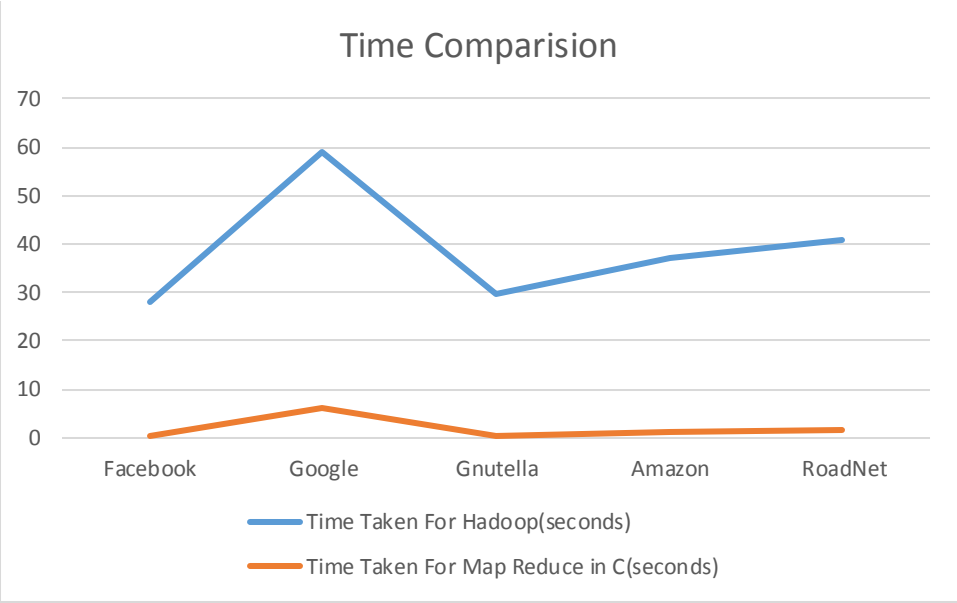
For RoadNet Graph,

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 35.83      0.05     0.05  1011265     0.00     0.00  simulateMapperFunction
 28.66      0.09     0.04        1    40.13    40.13  initializeMapperDataStructure
 21.50      0.12     0.03        1    30.09    30.09  ReduceFunction
  7.17      0.13     0.01        1    10.03   110.34  MapperFunction
  7.17      0.14     0.01        1    10.03    10.03  storeIntermediateResults
  0.00      0.14     0.00        1     0.00     0.00  initializeDegreeArray
  0.00      0.14     0.00        1     0.00     0.00  initializeVerticesAndEdges
  0.00      0.14     0.00        1     0.00     0.00  outputResults


granularity: each sample hit covers 2 byte(s) for 7.12% of 0.14 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00    0.14                 main [1]
                0.01    0.10       1/1           MapperFunction [2]
                0.03    0.00       1/1           ReduceFunction [5]
-----------------------------------------------
                0.01    0.10       1/1           main [1]
[2]     78.6    0.01    0.10       1         MapperFunction [2]
                0.05    0.00 1011265/1011265     simulateMapperFunction [3]
                0.04    0.00       1/1           initializeMapperDataStructure [4]
                0.01    0.00       1/1           storeIntermediateResults [6]
                0.00    0.00       1/1           initializeVerticesAndEdges [8]
-----------------------------------------------
                0.05    0.00 1011265/1011265     MapperFunction [2]
[3]     35.7    0.05    0.00 1011265         simulateMapperFunction [3]
-----------------------------------------------
                0.04    0.00       1/1           MapperFunction [2]
[4]     28.6    0.04    0.00       1         initializeMapperDataStructure [4]
-----------------------------------------------
                0.03    0.00       1/1           main [1]
[5]     21.4    0.03    0.00       1         ReduceFunction [5]
                0.00    0.00       1/1           initializeDegreeArray [7]
                0.00    0.00       1/1           outputResults [9]
-----------------------------------------------
                0.01    0.00       1/1           MapperFunction [2]
[6]      7.1    0.01    0.00       1         storeIntermediateResults [6]
-----------------------------------------------
                0.00    0.00       1/1           ReduceFunction [5]
[7]      0.0    0.00    0.00       1         initializeDegreeArray [7]
-----------------------------------------------
                0.00    0.00       1/1           MapperFunction [2]
[8]      0.0    0.00    0.00       1         initializeVerticesAndEdges [8]
-----------------------------------------------
                0.00    0.00       1/1           ReduceFunction [5]
[9]      0.0    0.00    0.00       1         outputResults [9]
-----------------------------------------------
```

Problem 2: The goal of this problem is to make sure you know how to write the map/reduce-type distributed graph mining program. You are asked to adopt the Java Hadoop program for word counts across a collection of documents provided with the Hadoop tutorial (Part 3) to the program that computes the degree of each vertex in the graph. The file format for each graph is the same as in Problem #1.

(1) Compare the timing your Hadoop program takes using a single mapper and a single reducer versus the C++/C program in Problem #1. Do you see the differences in performance? What factors may contribute to such differences? (5 pts)

Graph Type	Time Taken For Hadoop(seconds)	Time Taken For Map Reduce in C(seconds)
Facebook	28.164	0.118
Google	59.134	6.193
Gnutella	29.75	0.209
Amazon	36.964	1.306
RoadNet	40.933	1.489



Note: Time taken is calculated using time command of Linux operating system.
Eg: time hadoop jar MapReduceDegreeGraphApplication-0.0.1.jar /facebook /facebook-output

Observation: We see that all the map reduce operations in C to calculate the degree of a graph takes less time than the actual map reduce operations in Hadoop for single cluster. Since Hadoop is based on the idea of distributed computing for multiple nodes for a large data set, it will give us a great computational abilities in the case where we cannot run them on a single machine because of the size of the data and we run of multiple nodes. In these cases, if we used a multiple node cluster to

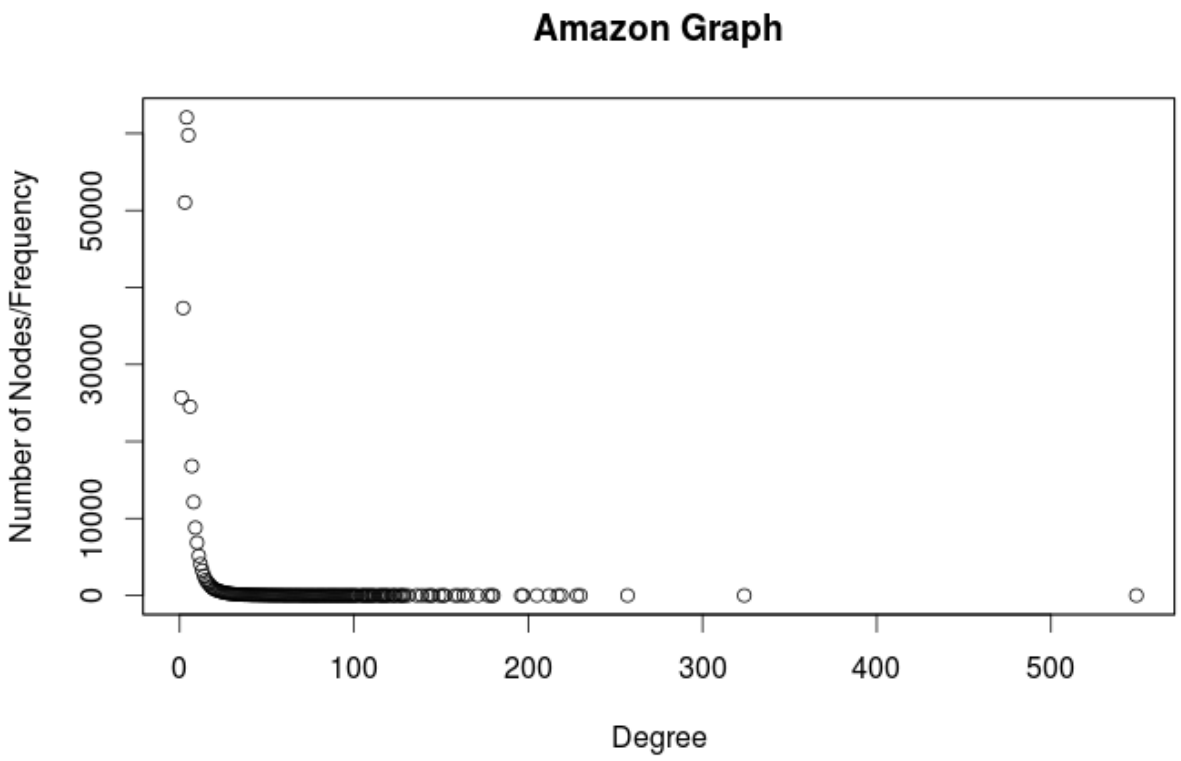
compute the degree in parallel fashion, we will be able to see a great time gain. For a single node cluster, it is not beneficial since we are over engineering the solution and not using map reduce paradigm for what it's known. Also because C code runs faster as compared to Java and also since C program written for map reduce is pretty straight forward as compared to Hadoop on single node cluster we see faster results for C program.

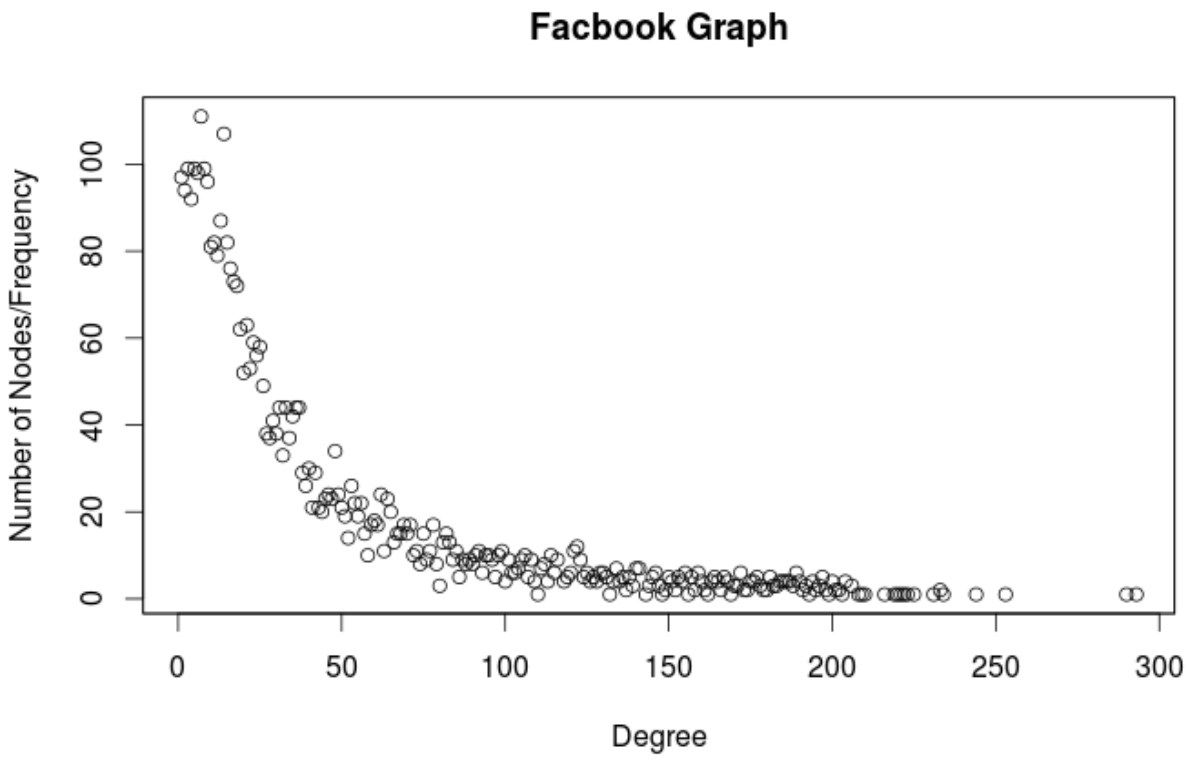
Problem 3: The goal of this problem is to make sure how you could compose more sophisticated analytical pipelines and apply them to large-scale real-world graphs such as those from Amazon, Wiki, Facebook, etc. Specifically, you are asked to expand the solution to Problem #2 from calculating the degree of each vertex to calculating the FREQUENCY of vertex degree (or degree distribution), namely, the number of vertices of a certain degree.

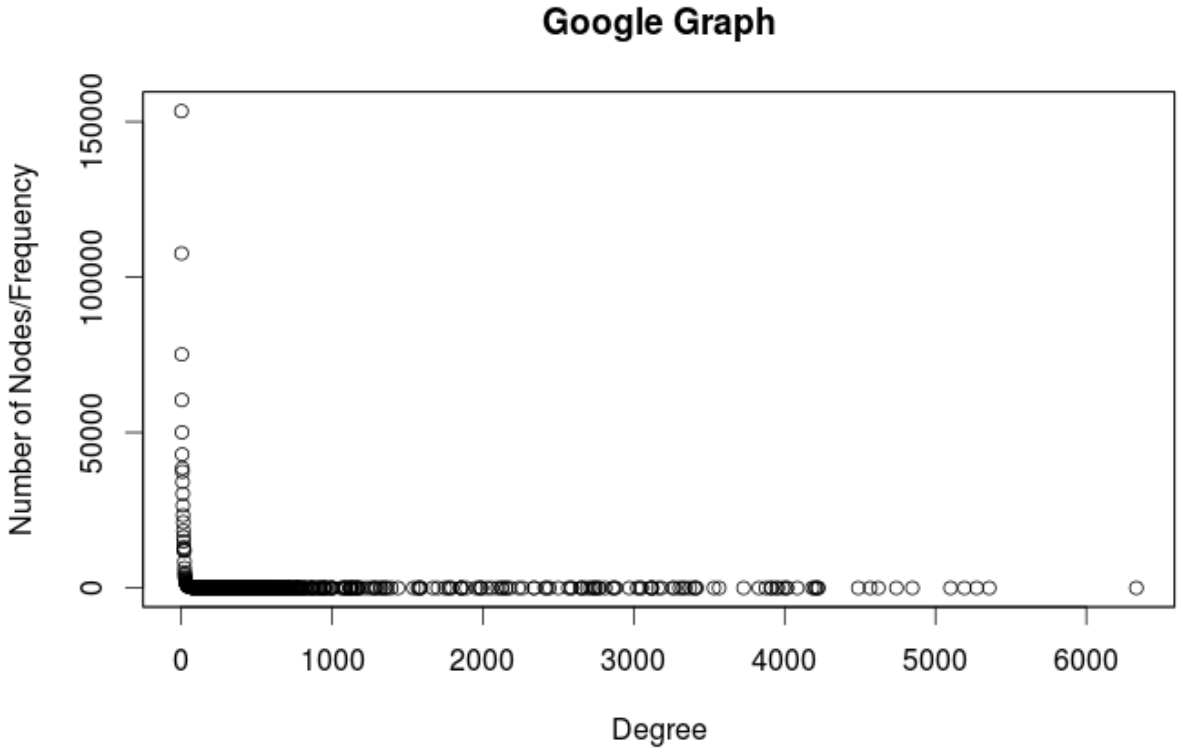
(1) For the Stanford graphs provided to you, test and report which graphs are scale-free, namely whose degree distribution follows a power law, at least asymptotically. That is, the fraction $P(k)$ of nodes in the network having k connections to other nodes goes for large values of k as

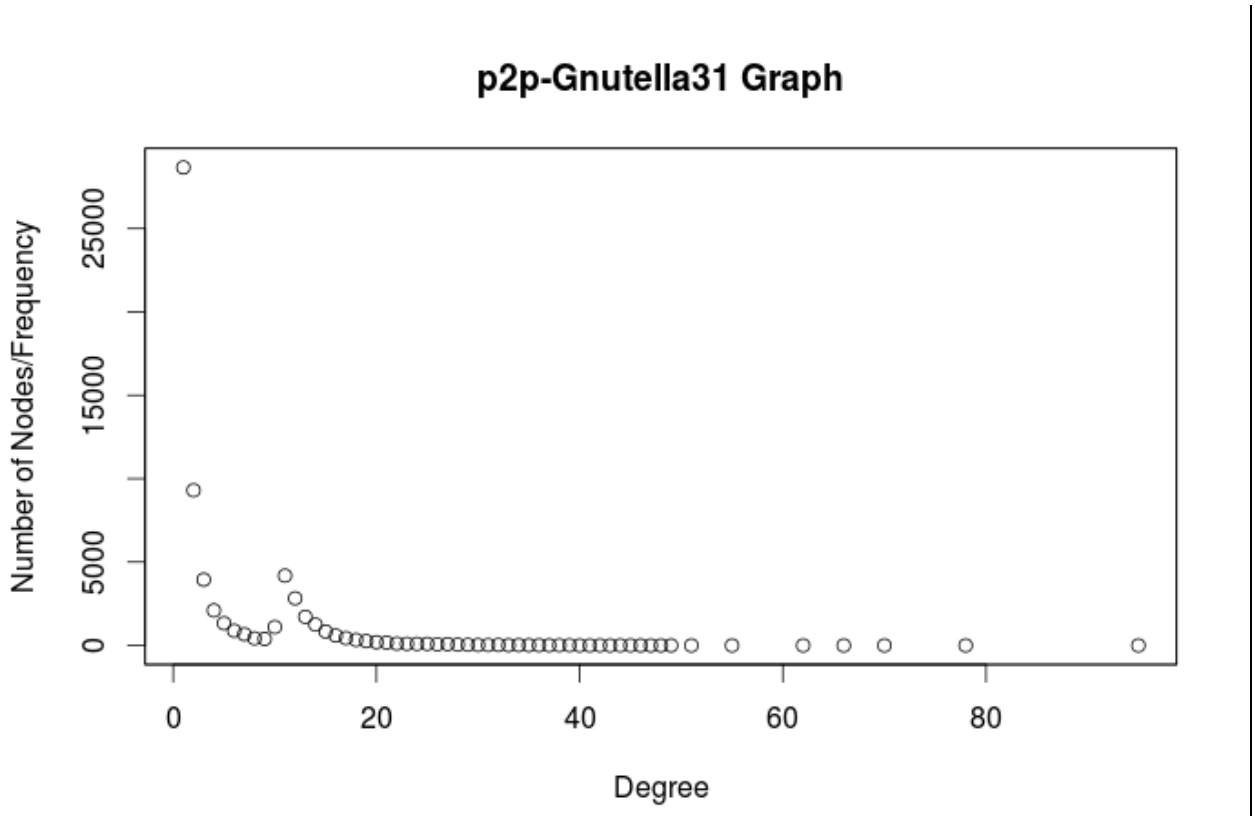
$$P(k) \sim k^{-\gamma}$$

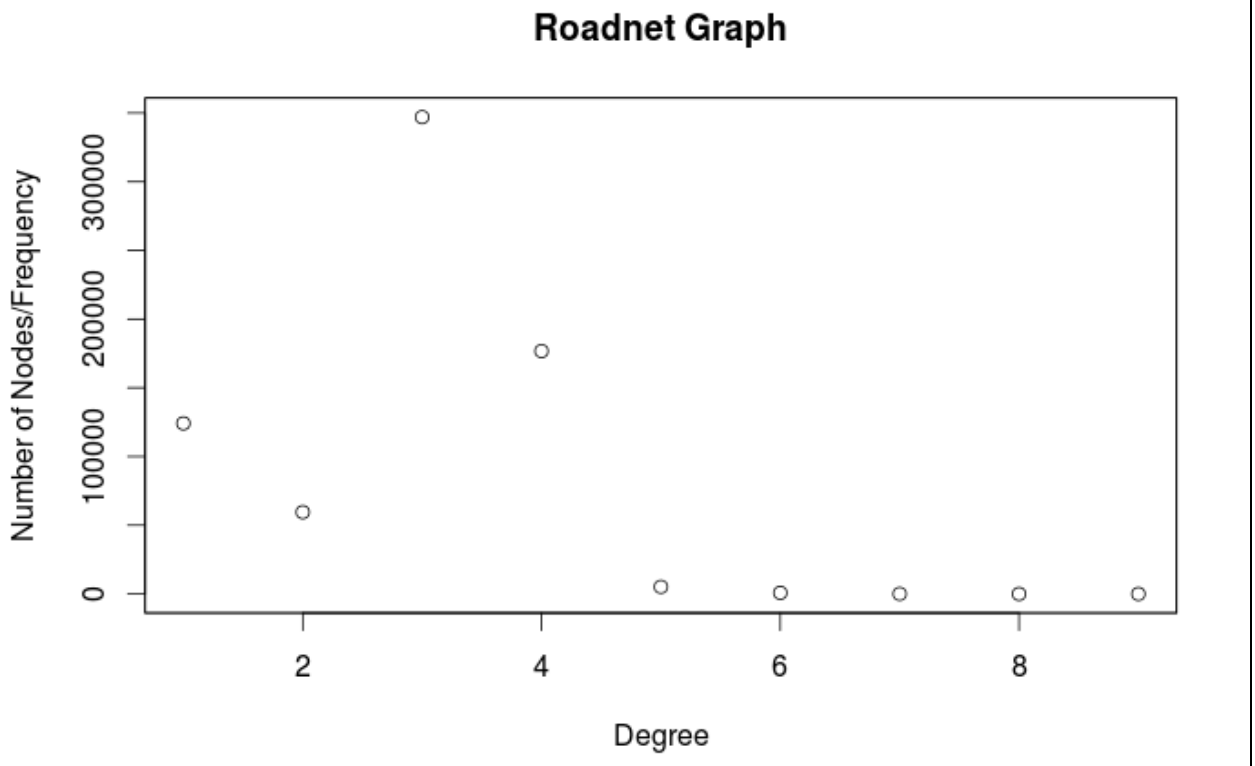
where ∨ is a parameter whose value is typically in the range $2 < \gamma < 3$, although occasionally it may lie outside these bounds.




Facbook Graph
Number of Nodes/Frequency
0
20
40
60
80
100
0
50
100
150
200
250
300
Degree


Google Graph
Number of Nodes/Frequency
0
50000
100000
150000
0
1000
2000
3000
4000
5000
6000
Degree

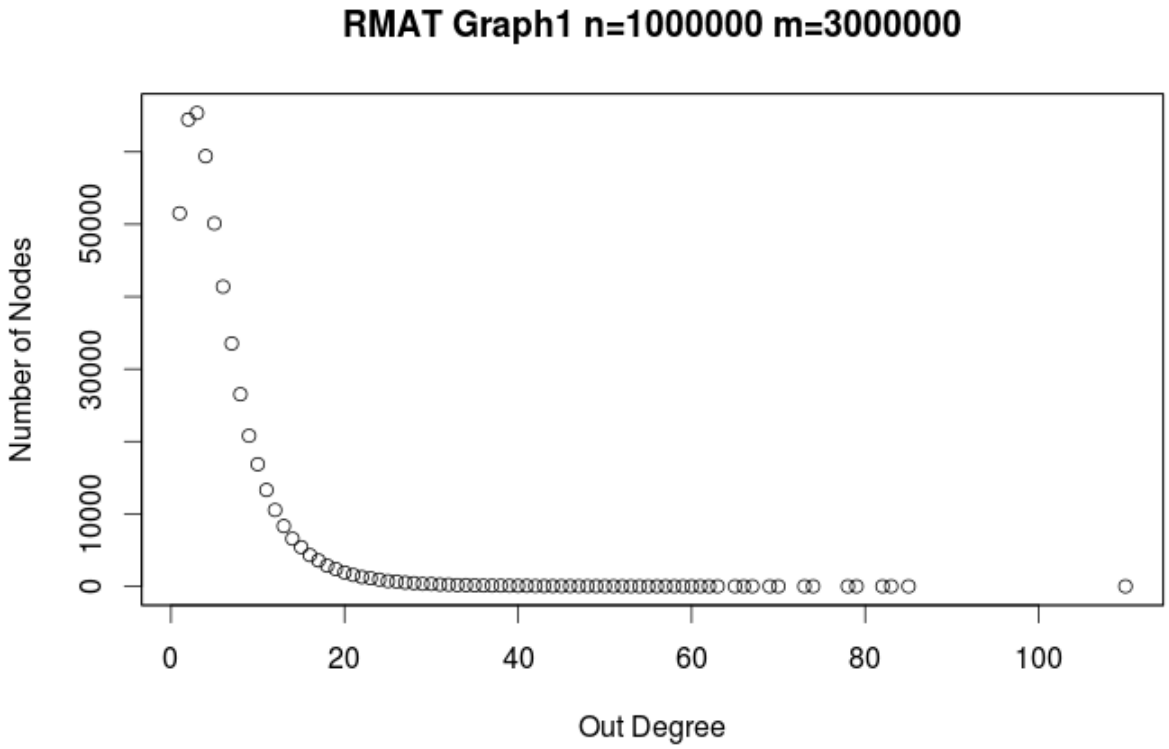

p2p-Gnutella31 Graph
Number of Nodes/Frequency
0
5000
15000
25000
0
20
40
60
80
Degree

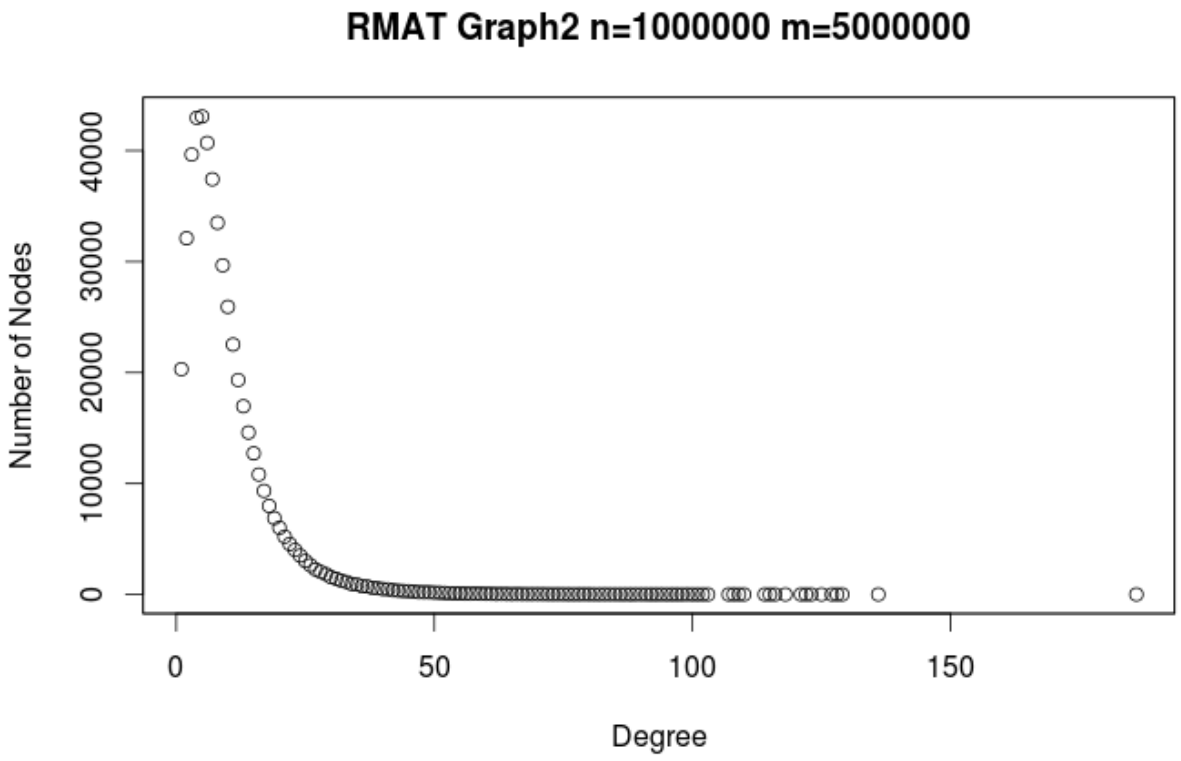

Roadnet Graph
Number of Nodes/Frequency
0
100000
200000
300000
2
4
6
8
Degree

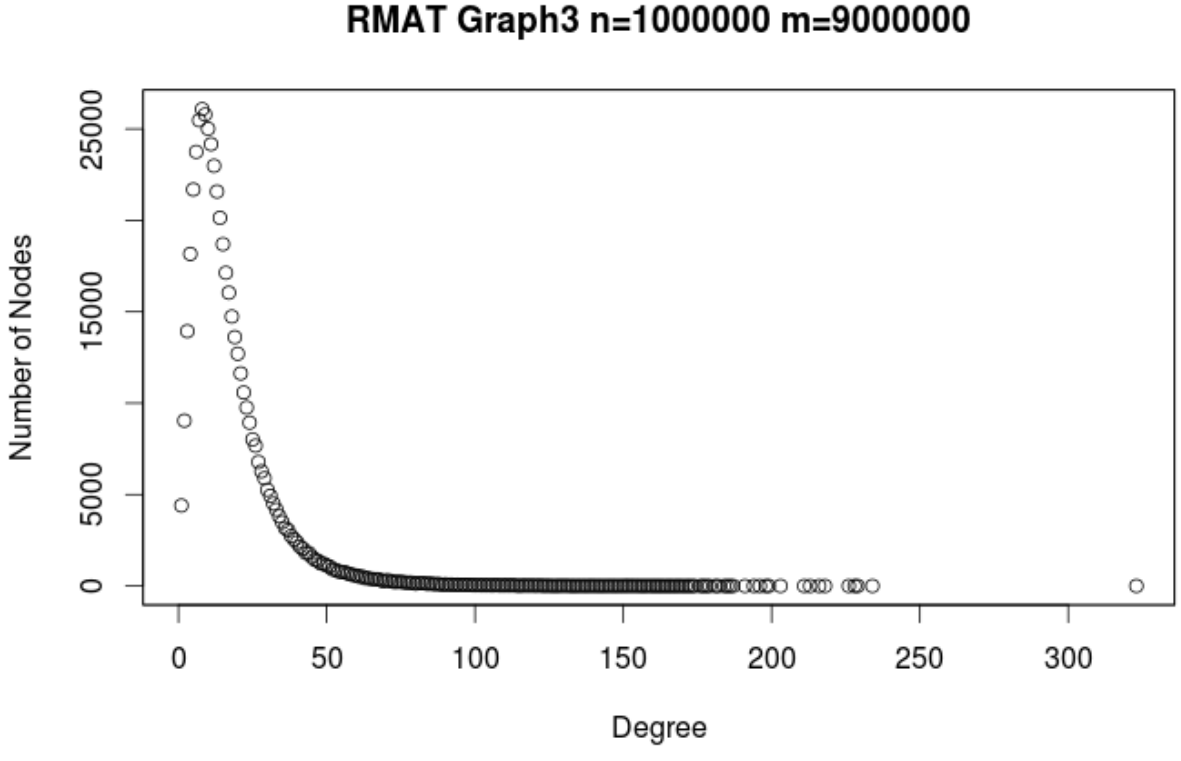
Observation:

- We see that Amazon and Google graphs perfectly adhere to power law looking at the plot since they follow the distribution $P(k) \sim k^{-\gamma}$. Hence they are scale free graphs.
- Facebook Graph and p2p-Gnutella graph has few outliers where the values do not exactly fall under the curve where $2 <= \gamma <= 3$ when values are normalized. However we can say that they are close to being scale free graph relaxing the outliers.
- RoadNet has two outlier points out of nine total points deviating way too much from the scale free distribution .Hence we cannot classify it as scale free because of two points(20% of points) are outside of scale free curve. If those two points are ignored then this can be considered to be scale free.

(2) Generate one million node random graphs **(multiple)** using R-MAT graph generator with default parameters (see, announcement 'VCL Image for Project 1' for location of R-MAT). Are those generated graphs scale-free?




RMAT Graph2 n=1000000 m=5000000
Number of Nodes
0
10000
20000
30000
40000
0
50
100
150
Degree


RMAT Graph3 n=1000000 m=9000000
Number of Nodes
0
5000
15000
25000
0
50
100
150
200
250
300
Degree

Observations: I generated three RMAT graphs for various configurations of n(vertices) and m(edges) and found that all the graphs adhere to power law making **them all scale free graphs**. Please see the Output folder for all the outputs generated.