Carnegie Mellon University
15-826 – Multimedia Databases and Data Mining
Fall 2013, C. Faloutsos
Homework 2, Due Date: Nov 5th, at classroom 1:30pm
Prepared by: Alex Beutel (Q4, Q5) & Vagelis Papalexakis (Q1, Q2, Q3)

Reminders

- All homeworks are to be done **INDIVIDUALLY**.
- All written answers should be **TYPED**.
- For code submission to blackboard, please follow the template of the `hw.zip` file that you can find download from `http://www.cs.cmu.edu/~christos/courses/826.F13/HOMEWORKS/HW2/hw2.zip`. You should rename the final .zip file to [andrew-id].zip.
- FYI: Expected effort for this homework (approximate times):
  - Q1: 1-2 hours (**To be graded by Seunghak Lee**)
    - 1-2 hours to solve and write it up.
  - Q2: 4 hours (**To be graded by Vagelis Papalexakis**)
    - 1-2 hours to download the datasets and the software package.
    - 0.5-1 hours to see how the software works.
    - 1-2 hours to generate the plots and write up the solution.
  - Q3: 10 hours (**To be graded by Vagelis Papalexakis**)
    - 3-4 hours to write the code for the 80/20 multifractal.
    - 2-3 hours to modify the above code for the 70/20/10 multifractal.
    - 1 hour to generate the correlation integral plots using FDNQ.
    - 2-3 hours to code up the exact correlation integral and finalize the rest of the question.
  - Q4: 20 hours (**To be graded by Seunghak Lee**)
    - 15 hours to code the string edit distance script
    - 4 hours to test it for corner cases
    - 1 hour to calculate the answers for the question
  - Q5: 10 hours (**To be graded by Alex Beutel**)
    - 7 hours for the SQL statement
    - 1 hour for plotting the degree distribution
    - 2 hour for answering questions around speed

Q1 – Density Paradox [5 pts]

(On separate page)

Problem Description: In this problem we will see the density paradox. You are given N=1 million points on the 45-degree line (i.e., $x = y$ line), which starts from (-M, -M) and up to (M,M), where M is a large number.
Compute the density at the origin. That is, consider the $(0, 0)$ point; we are told that the number of its neighbours $NN(r)$ for radius $r = 1$ is $NN(1) = 20$. For simplicity, we consider the L_∞ norm, which means that we have a square of radius r (and thus, of side $2r$), centered at the origin, and within this square, there are 20 points.

1. [2.5 pts] Calculate the density for the following radii: [*HINT*: estimate the number of neighbours, for each radius, using the formula $NN(r) = C\ r^{D_2}$:
   - $r = 5$
   - $r = 10$
   - $r = 20$
2. [2.5 pts] What do you observe? Is the above question well posed?

What to turn in:

- **On Paper:** Please turn in the answers to the above question. *(On separate page)*
- **Online:** There is no need to submit something for this question.

Solution The way we calculate the density for a given radius r is $d(r) = \frac{NN(r)}{(2r)^2}$. For $r = 1$, we solve for C and we get $C = 20$.

- For $r = 5$,
$$d(5) = \frac{C5^1}{100} = \frac{20}{20} = 1$$
- For $r = 10$,
$$d(10) = \frac{10C}{400} = \frac{20}{40} = \frac{1}{2}$$
- For $r = 20$,
$$d(20) = \frac{20C}{1600} = \frac{20}{80} = \frac{1}{4}$$

We observe that the density is varying, according to the radius, which is not correct, since the density should be constant. Therefore, the question is not well posed, since calculating the density for a fractal object (and the line is trivially a fractal object), is an ill posed problem.

Q2 – Fractals [10 pts]

(On separate page)

Problem Description: In this problem you will experiment with Fractal dimension, and how to use it as a feature, in order to tell whether a dataset looks 'realistic' or not. You are given 5 different datasets with coordinates in the 2-d space. You can download an archive containing all these datasets from `http://www.cs.cmu.edu/~epapalex/15826F13/data/datasets.zip` [**CMU only access** - please make sure you download it using an SCS machine, or login via VPN]. Each dataset is formatted as:

x_coordinate y_coordinate

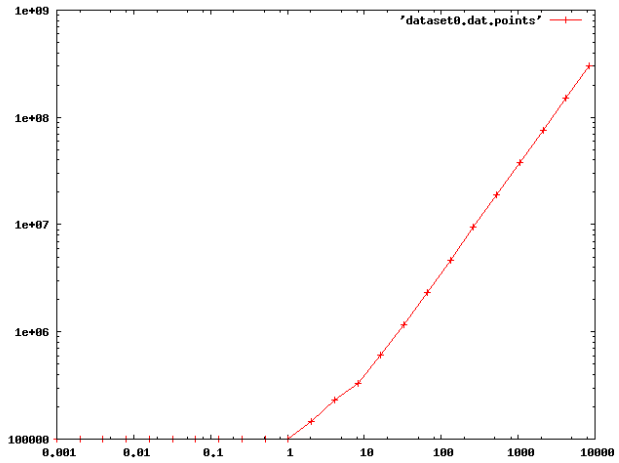
1. [5 pts] For each one of those datasets, *plot* the correlation integral. You may use the code of the FDNQ package `http://www.cs.cmu.edu/~christos/SRC/fdnq_h.zip`
2. [5 pts] *Using the information provided by the correlation integral*, find out which one(s) (if any) of the datasets are probably not real, and justify your solution based on the correlation integral. [HINT: The fractal dimension of the fake datasets will be close to integer]

What to turn in:

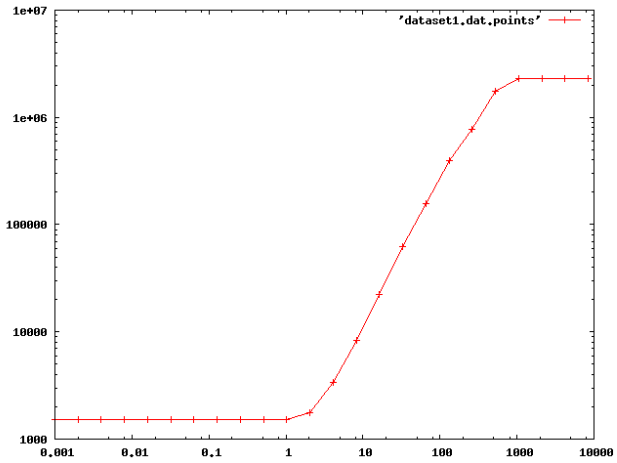
- **On Paper:** *(On separate page).* Please turn in printouts of the correlation integrals, the slope for each of the correlation integrals, as well as the numbers of the datasets that you identified as fake.
- **Online:** Please turn in the code that you used in order to generate the required plots, including the `run.sh` bash script we have given you in the homework template, after you fill in the calls to your code.In order to execute this script, you can type `bash run.sh`. You should also turn in the plots in .pdf form.

Solution

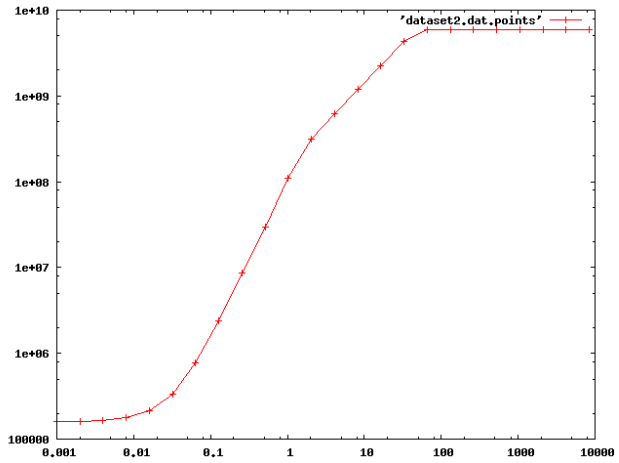
1. The correlation integrals, as computed by FDNQ, for the given datasets are:
2. Based on the slope of the correlation integral, Datasets 0 and 4 have slope close to 1, and therefore are the most likely to be fake. Indeed, dataset 0 consists of points on a line, and dataset 4 consists of points on a circle; both datasets where artificially generated.



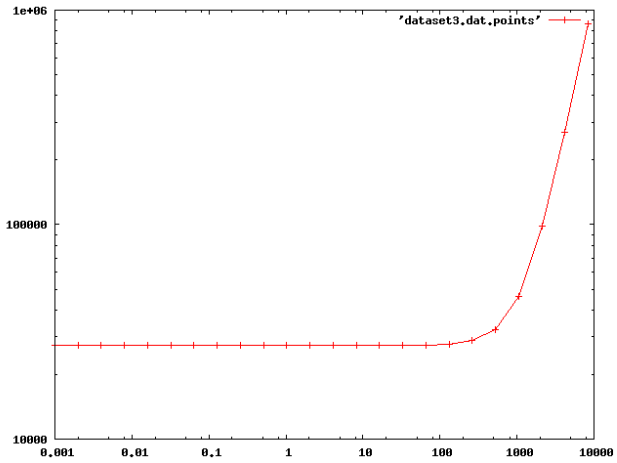
(a) Dataset 0, slope = 0.956681



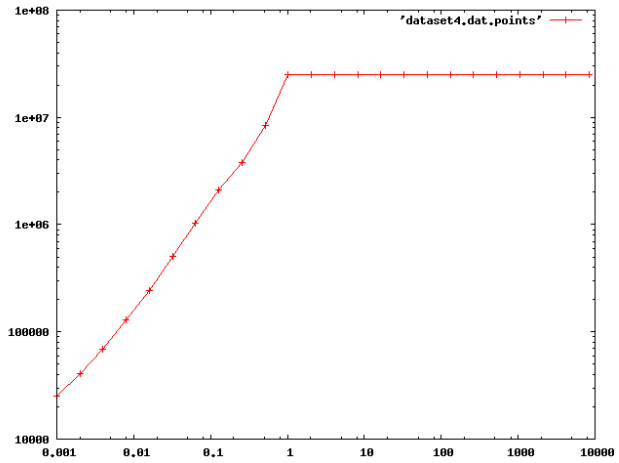
(b) Dataset 1, slope = 1.34278



(c) Dataset 2, slope = 1.49235



(d) Dataset 3, slope = 1.45772



(e) Dataset 4, slope = 0.988007

Figure 1:

Q3– Self Similar Time Sequence & Multifractals [25 pts]

(On separate page)

Problem Description: In this problem you will experiment with Multifractals, and the generation of bursty, realistic, self-similar time sequences, as we saw in class. Each value of the histogram you will create corresponds to, say, the number of disk accesses at a time interval.

1. [7 pts] Write code that generates and plots a multifractal bursty time-sequence, using the 80/20 rule. As we said in the course foils, a bias b=0.8 means that within a given time interval, 80% of the accesses happen in the right half of the interval and the remaining 20% in the left half; and this splitting of accesses happens recursively for each of those halves. You should use 2048 time-ticks, and 2048 disk accesses. [*HINT*: See [Wang et al.] `http://www.pdl.cs.cmu.edu/PDL-FTP/Workload/CMU-CS-01-101.pdf`]

2. [7 pts] Write code that generates and plots a multifractal bursty time-sequence, using the *70/20/10* rule. That is, instead of splitting each time interval into two parts, we split it into *three*; then, with probability 70% we are generating an access on the first third, 20% on the second third, and 10% on the last third. Use the same configuration for the disk accesses, as well as the same convention for choosing the segments (70% right, 20% middle, 10% left). For the number of time-ticks, you have to choose a power of 3, so pick 2187.

3. [5 pts] Plot the correlation integral for the time sequences you generated, using the FDNQ package `http://www.cs.cmu.edu/~christos/SRC/fdnq_h.zip`. What is the slope?

4. [5 pts] Write code that computes and plots the correlation integral using the *exact*, quadratic method that we saw in class. You may use your favourite language, as long as it compiles on an Andrew machine. What is the slope? Does it differ significantly from the one you found using FDNQ?

5. [1 pt] Using the correlation integral and its slope, verify that the fractal dimension D for the $b = 80$ multifractal is equal to

$$D = -log_2(b^2 + (1-b)^2)$$

[HINT: It is fine if the two numbers are approximately equal.]

What to turn in:

- **On Paper:** *(On separate page).* Please turn in (a) the numerical answers to all questions above, (b) the printouts of all the plots and (c) a printout of the code that *you* wrote.

- **Online:** Please turn in the code that you wrote/used in order to generate the required plots including the `run.sh` bash script we have given you in the homework template, after you fill in the calls to your code. In order to execute this script, you can type `bash run.sh`. You should also turn in the plots in .pdf form.

Solution

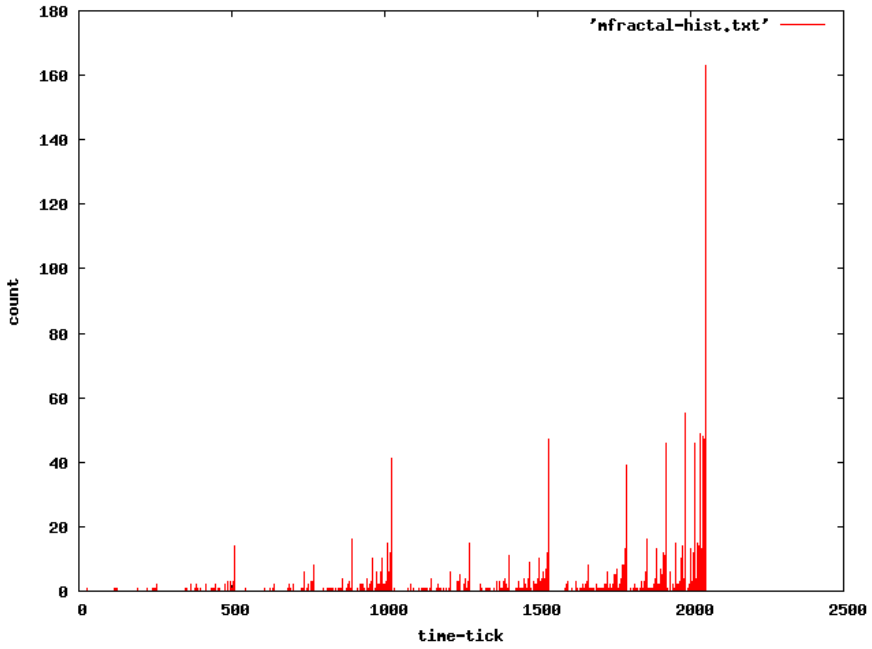
1. The Python code for the multifractal generation is:

```
N = 2048    # number of disk accesses
k = 11       # 2**k = number of buckets = time-ticks
p = 0.8     # bias factor
s = 1       # seed - set it, for repeatability


import getopt
import sys
import random


random.seed(s)

for i in range(N):
    # print i
    value = 0
    # generate k bits, with biased probability
    # Their decimal value is the bucket-id = time-tick
    for pos in range(k):
        r = random.random()
        if(r < p):
            bit_value = 1
        else:
            bit_value = 0
        value = bit_value + 2*value
    print value
```

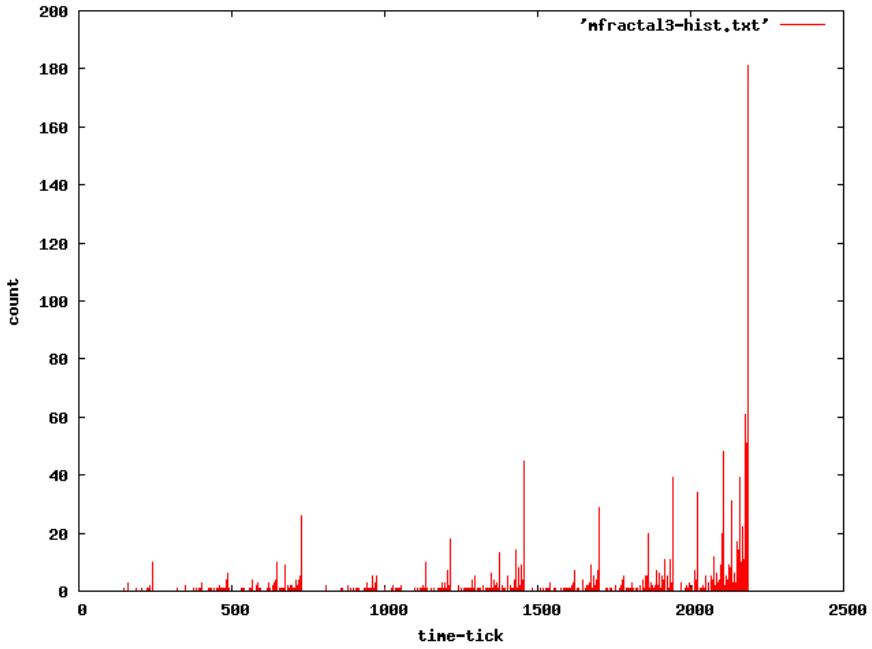
   The disk accesses generated by the 80/20 multifractal are shown in Fig

2. The Python code for the 3-multifractal generation is:

```
N = 2048    # number of disk accesses
k = 7       # 3**k = number of buckets = time-ticks
p1 = 0.7     # bias factor
p2 = 0.2
```



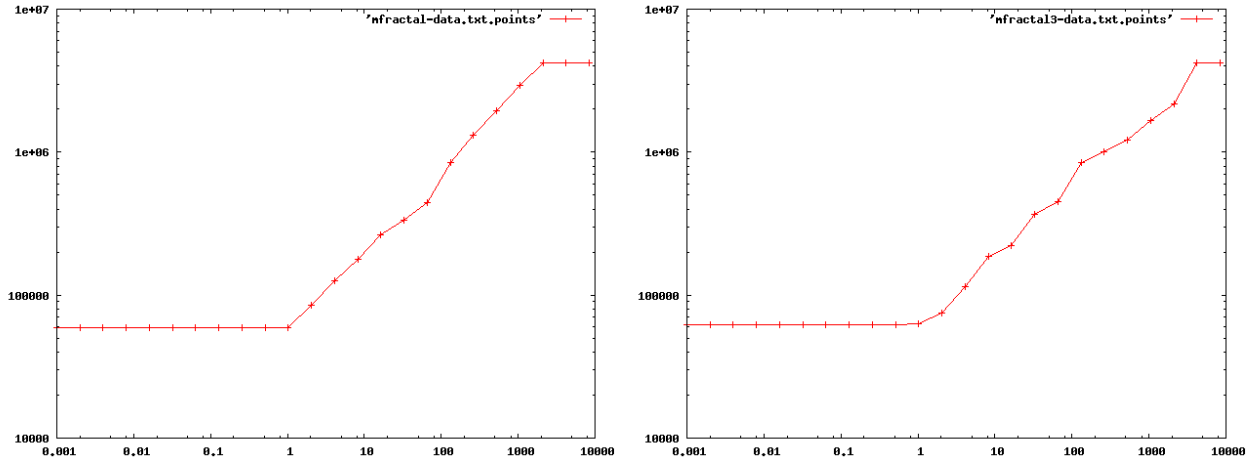

```
p3 = 0.1
s = 1       # seed - set it, for repeatability


import getopt
import sys
import random


random.seed(s)

for i in range(N):
    # print i
    value = 0
    # generate k bits, with biased probability
    # Their decimal value is the bucket-id = time-tick
    for pos in range(k):
        r = random.random()
bit_value = 0
        if(r < p1):
            bit_value = 2
        elif(r<(p1+p2)):
            bit_value = 1
```




```
    value = bit_value + 3*value
print value
```

The disk accesses generated by the 70/20/10 multifractal are shown in Fig

3. The correlation integrals for the Multifractal and the 3-Multifractal, as computed by FDNQ (and their respectful slopes) are:

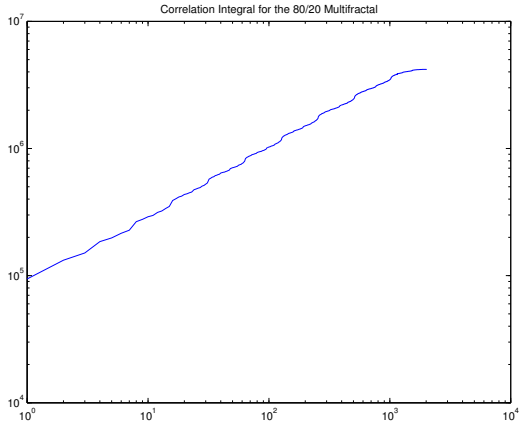


(a) 80/20 Multifractal (FDNQ), slope= 0.565458 (b) 70/20/10 Multifractal (FDNQ), slope= 0.472779

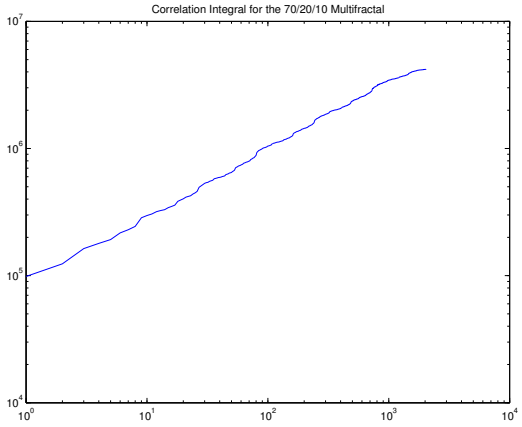
Figure 2:

4.
```
function [output,allr] = correlation_integral(x)

%First find all possible radii
allr = zeros(length(x)^2,1);
idx = 1;
for m = 1:length(x);
    for n = 1:length(x)
        allr(idx) = sqrt(sum((x(m) - x(n)) .^ 2));
        idx = idx+1;
    end
end

allr = unique(allr);
allr = sort(allr,'ascend');%sort radii in ascending order

%count all the pairs that are within distance <= a given radius
output = zeros(size(allr));

for i = 1:length(allr)
    r = allr(i);
    for m = 1:length(x)
        for n = 1:length(x)
            if m == n
                continue
            end
            distance = abs(x(m) - x(n));
            if(distance<=r)
               output(i) = output(i) + 1;
            end
        end
    end

end


p = polyfit(log10(x),log10(y),1);
slope = p(1)
```

5. If we substitute the numbers in the formula, we get a fractal dimension of 0.5564 which is approximately equal to the one we estimate from the generated data points.



(a) 80/20 Multifractal (quadratic algorithm), slope= 0.4988



(b) 70/20/10 Multifractal (quadratic algorithm), slope= 0.5003

Figure 3:

Q4– String Editing Distance [30 pts]

(On separate page)

Problem Description: Write a program to compute the string edit distance and the path that the string editing takes, as can be seen in the example below.
Compute the following:

1. [15 pts] Compute the string edit distance between the two strings below:

   `The quick brown fox jumps over the lazy dog`

   and

   `Das quik brown foxxx jumps over the lay-z dogg`

   Use a deletion/insertion cost of 1 and a substitution cost of 0.5.

2. [15 pts] Run the same command with deletion/insertion cost of 0.5 and substitution cost of 5.

Example output: For "`some string`" and "`somestrrng 2`" with weights 1 and 0.5:

```
Cost: 3.5
s -> s
o -> o
```

```
m -> m
e -> e
  -> *
s -> s
t -> t
r -> r
i -> r
n -> n
g -> g
* ->
* -> 2
```

Note: Use an asterisk * as a placeholder when there is an insertion or deletion as in the example above.

What to turn in:

- **On Paper:** *(On separate page).* Please turn in the cost and the "path" that the minimum string edit distance takes as seen in the example above. Also turn *all* code you used to generate your output.
- **Online:** Please turn in your code as well as a bash script named `stredit.sh` such that the command:
  `bash stredit.sh "some string" "somestrrng 2" 1 0.5`
  will generate the desired output as in the example above. A template can be found at `http://www.cs.cmu.edu/~christos/courses/826.F13/HOMEWORKS/HW2/Q4/q4.tar.gz`

  Note, if your program requires compiling please submit the source code and have your bash script compile your code before running it.

Solution
The paths for the two questions are given below. Overall we see that the cost in the first case is 7.5, and in the second case is 6.5. The python code can be found below the paths and downloaded from `http://alexbeutel.com/edit_dist.py`.

1. Deletion = 1 and substitution = 0.5

   ```
   Cost: 7.5
   T -> D
   h -> a
   e -> s
     ->
   q -> q
   u -> u
   ```

```
i -> i
c -> *
k -> k
  ->
b -> b
r -> r
o -> o
w -> w
n -> n
  ->
f -> f
o -> o
x -> x
* -> x
* -> x
  ->
j -> j
u -> u
m -> m
p -> p
s -> s
  ->
o -> o
v -> v
e -> e
r -> r
  ->
t -> t
h -> h
e -> e
  ->
l -> l
a -> a
z -> y
y -> -
* -> z
  ->
d -> d
o -> o
g -> g
* -> g
```

2. Deletion = 0.5 and substitution = 5

```
Cost: 6.5
* -> D
* -> a
* -> s
T -> *
h -> *
e -> *
  ->
q -> q
u -> u
i -> i
c -> *
k -> k
  ->
b -> b
r -> r
o -> o
w -> w
n -> n
  ->
f -> f
o -> o
x -> x
* -> x
* -> x
  ->
j -> j
u -> u
m -> m
p -> p
s -> s
  ->
o -> o
v -> v
e -> e
r -> r
  ->
t -> t
h -> h
e -> e
  ->
l -> l
```

```
    a -> a
    * -> y
    * -> -
    z -> z
    y -> *
      ->
    d -> d
    o -> o
    g -> g
    * -> g

import sys
import os

def main():
        if (len(sys.argv) >= 5):
                word1 = sys.argv[1]
                word2 = sys.argv[2]
                deletion_cost = float(sys.argv[3])
                sub_cost = float(sys.argv[4])
        else:
                print "No parameters given. Use format: "
                print 'python edit_dist.py  "string 1" "string 2" 1 0.5'
                print 'where 1 is the example deletion cost and 0.5 is the ex
                print "\nRunning in demo mode with 'some string' and 'somestr
                word1 = "some string"
                word2 = "somestring 2"
                deletion_cost = 1
                sub_cost = 1

        len1 = len(word1) + 1
        len2 = len(word2) + 1

        DP = [[0 for x in xrange(len2)] for x in xrange(len1)]
        path = [[(0,0) for x in xrange(len2)] for x in xrange(len1)]

        for i in range(0,len1):
                DP[i][0] = i * deletion_cost
                if i > 0:
                        path[i][0] = (-1,0)

        for j in range(0,len2):
                DP[0][j] = j * deletion_cost
```

```
        if j > 0:
                path[0][j] = (0,-1)

for i in range(1,len1):
        for j in range(1,len2):
                cost1 = DP[i-1][j] + deletion_cost
                cost2 = DP[i][j-1] + deletion_cost

                sc = sub_cost
                if ( word1[i-1] == word2[j-1] ):
                        sc = 0

                sc = sc + DP[i-1][j-1]

                DP[i][j] = min(cost1,cost2,sc)
                if( cost1 == DP[i][j]):
                        path[i][j] = (-1,0)
                elif (cost2 == DP[i][j]):
                        path[i][j] = (0,-1)
                else:
                        path[i][j] = (-1,-1)


### Uncomment to view matrices of costs and paths
#for i in range(0,len(DP)):
        #print DP[i]
#for i in range(0,len(path)):
        #print path[i]

print "Cost: " + str(DP[len1-1][len2-1])

i = len1 -1
j = len2 -1
sol = ''
while True:
        inew = i + path[i][j][0]
        jnew = j + path[i][j][1]

        ### Uncomment to watch path unroll
        #print str(inew) + ", " + str(jnew)
        w1 = '*'
        w2 = '*'
```

```
                if i != inew:
                        w1 = word1[inew]

                if j != jnew:
                        w2 = word2[jnew]

                sol = w1 + " -> " + w2 + "\n" + sol

                i = inew
                j = jnew
                if inew == 0 and jnew == 0:
                        break

        print sol


if __name__ == '__main__':
        main()
```

Q5– Graph Mining and Anomaly Detection [30 pts]

(On separate page)
Here you will learn to mine data from the graph structure. The directed graph you will be operating can be downloaded from `http://cs.cmu.edu/~abeutel/patents.csv`. In this file each line is an edge specified as `fromNode,toNode`. Please complete the following tasks using SQL.

Problem Description:

1. [3 pts] How many nodes are there in the graph? (Denote this as N.)
2. [3 pts] What are the total number of edges (or 1 hop away neighbors) in the graph? How long does it take to find this value?
3. [5 pts] Find and plot the out degree distribution for the graph. Give the plot.
4. [3 pts] At what degree does there appear to be a surprising number of nodes with that out degree?

   For the following questions, consider the graph to be undirected. The undirected dataset can be downloaded from `http://cs.cmu.edu/~abeutel/undirected_patents.csv`:

5. [3 pts] What is the average degree of each node? (Denote this as d_{avg}.)
6. [3 pts] What is the maximum degree? (Denote this as d_{max}.)
7. [3 pts] **(Note: If your query takes over an hour for this question, stop it.)** What is the total number of 2 step away neighbors for all nodes?

   Also, answer the following related questions:

   (a) [4 pts] Select which of the below statements about the total number of 2 step away neighbors are true.
      i. The value is approximately $N \cdot d_{\text{avg}} \cdot d_{\text{avg}}$ because everybody has approximately d_{avg} neighbors who have approximately d_{avg} neighbors.
      ii. The value is more than $N \cdot d_{\text{avg}}^2$
      iii. The value is at most $\sum_{i=1}^{N} d_i^2$ where d_i is the degree of node i.
      iv. The value is at least $\sum_{i=1}^{N} d_i^2$ where d_i is the degree of node i.
      v. The value is more than d_{max}^2

   (b) [3 pts] Calculate the values of the following using SQL:
      i. $N \cdot d_{\text{avg}}^2$
      ii. $\sum_{i=1}^{N} d_i^2$
      iii. d_{max}^2

What to turn in:

- **On Paper:** *(On separate page).* Please turn in the answers to all questions above as well as all SQL commands needed for questions #1,2,3,5,6,7.
- **Online:** Please turn in your SQL code and a bash script to generate the answers to all of the questions above. You can follow the template bash script from `http://www.cs.cmu.edu/~christos/courses/826.F13/HOMEWORKS/HW2/Q5/q5.tar.gz`

Solution
To set up the database in sqlite3, we can use the following commands

```
create table patents(fromPatent TEXT, toPatent TEXT);
.separator ","
.import patents.csv patents

create table patents2(fromPatent TEXT, toPatent TEXT);
.separator ","
.import undirected_patents.csv patents2
```

1. To get the number of unique nodes in the graph, use the following command

   ```
   SELECT COUNT(DISTINCT(node)) FROM
   (SELECT fromPatent as node FROM patents
   UNION
   SELECT toPatent as node FROM patents) as t1;
   ```

   The number of nodes is $N =$ **3784768**.

2. To get the number of edges in the graph or the number of one hop away neighbors we can use the following command:

   ```
   SELECT COUNT(*) FROM patents;
   ```

   The result is **16978948** .

3. The degree distribution can be found with

   ```
   SELECT degree, COUNT(*) FROM
   (SELECT fromPatent, COUNT(*) degree FROM patents GROUP BY fromPatent) degrees
   GROUP BY degree ORDER BY degree ASC;
   ```

   The degree distribution is seen below in Figure 4.

4. We see that there are a surprising number of nodes with an out degree of **46** (approximately 10,000 compared to the expected 800).

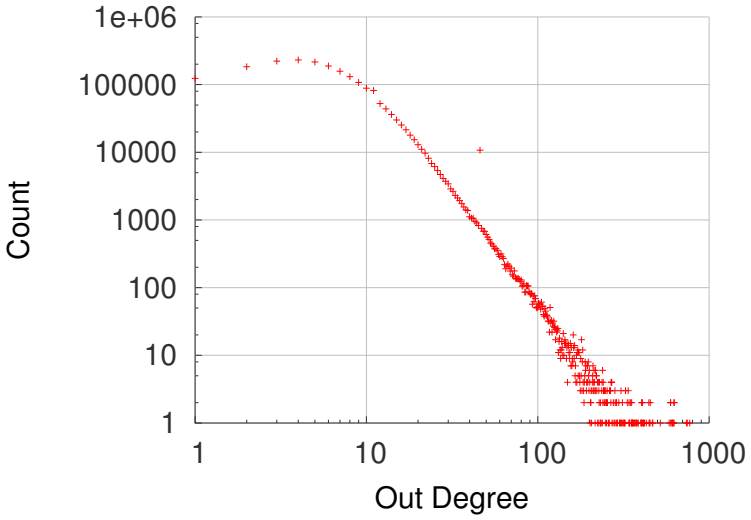


Figure 4: Out degree distribution of the modified patent graph.

5. The average degree can be found with

```
SELECT AVG(degree) FROM
(SELECT fromPatent, COUNT(*) degree FROM patents2 GROUP BY fromPatent) degrees
```

   The result is $d_{\text{avg}} = \mathbf{8.97225298882256}$.

6. The max degree can be foudn with

```
SELECT fromPatent, COUNT(*) degree FROM patents2
GROUP BY fromPatent ORDER BY degree DESC LIMIT 1
```

   The result is $d_{\text{max}} = \mathbf{10043}$.

7. The two hop away neighbors can be enumerated and counted with

```
SELECT COUNT(*) FROM
(SELECT p.fromPatent, p2.toPatent FROM
patents2 p JOIN patents2 p2 ON p.toPatent = p2.fromPatent
GROUP BY p.fromPatent, p2.toPatent) as twohops;
```

   However, even making an index this is very slow because of *how many* 2-hop away neighbors there are.

   (a) The true statements are **ii**, **iii**, and **v**. Overall we see that **iii** is a good approximation, however it does not take into account duplicates which is why it is an

upperbound. Because, we see that the degree of each node is squared (approximately), squaring the average, particularly for a power law distribution, gives an answer far too low. Similarly, it is clear that $d_{\max}^2$ is far too low since it only looks at the contribution of the node with the biggest degree.

(b) i. We have already calculated the values above for N and d_{avg} so we see that

$$N \cdot d_{\text{avg}}^2 = 3784768 \cdot 8.97225298882256^2 = 304678833.8801255$$

ii. To calculate this we must use SQL directly:

```
SELECT SUM(degree*degree) FROM
(SELECT fromPatent, COUNT(*) degree FROM patents2 GROUP BY fromPatent) d;
```

We find that the answer is $\sum_i d_i^2 = 5330860456$.

iii. We have already calculated $d_{\max}$ above so we can quicly find that $d_{\max}^2 = 100861849$.