1 Finding Trends

1.1

In [24]:

```
# Read txt
with open("test_set_tweets.txt","r", encoding='utf-8') as file:
    lines = [next(file) for x in range(500000)]
```

In [9]:

```
import re

def extractHashtags(string):
    pattern = re.compile(r"#(\S+)")
    strs = re.findall(pattern, string)

    pattern = re.compile('[^a-zA-Z]')
    output = []
    for i in strs:
        output.append(pattern.sub('', i.lower()))
    return output

# Example:
extractHashtags("22077441       10470781081     #confession.    I can't live with my mama!!! Espe
cially if I don't have my own car!      2010-03-14 09:21:58")
```

Out[9]:

```
['confession']
```

In [10]:

```
def mapper_hashtags_line(line):
    words = extractHashtags(line)
    output = []
    for word in words:
        if word:
            output.append((word,1))
    return output

# Example:
mapper_hashtags_line("22077441  10470781081     #confession.    I can't live with my mama!!! Espe
cially if I don't have my own car!      2010-03-14 09:21:58")
```

Out[10]:

```
[('confession', 1)]
```

In [15]:

```
def mapper_hashtags(lines):
    output = []
    for line in lines:
        list = mapper_hashtags_line(line)
        if list:
            output += list
    return output

#Example:
test = ["#John. 2010", "#Jerry 2011", "#Tom 2012", "#Jerry 2013"]
mapper_hashtags(test)
```

Out[15]:

```
[('john', 1), ('jerry', 1), ('tom', 1), ('jerry', 1)]
```

In [16]:

```
def combiner_heshtags(mapper_output):
    groups = {} # group by key values
    for item in mapper_output:
        k = item[0]
        v = item[1]
        if k not in groups:
            groups[k] = [v]
        else:
            groups[k].append(v)
    return groups

#Example:
combiner_heshtags(mapper_hashtags(test))
```

Out[16]:

```
{'john': [1], 'jerry': [1, 1], 'tom': [1]}
```

In [17]:

```
def reducer_heshtags(keyWord, counts):
    return (keyWord, sum(counts))

reducer_heshtags( 'jerry',[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

Out[17]:

```
('jerry', 14)
```

In [25]:

```
def execute_heshtags(lines):
    groups = combiner_heshtags(mapper_hashtags(lines))
    output = [reducer_heshtags(k,v) for k,v in groups.items()]
    output.sort()
    return output

hashtags_freq = execute_heshtags(lines)
```

In [33]:

```
def Sort(orig):

    orig.sort(key = lambda x: x[1], reverse = True)
    return orig

print(Sort(hashtags_freq)[:10])
```

```
[('ff', 3581), ('nowplaying', 1809), ('fb', 1402), ('mm', 1029), ('fail', 686),
('random', 622), ('haiti', 591), ('shoutout', 529), ('followfriday', 457), ('music
monday', 452)]
```

In [35]:

```
import timeit

start = timeit.default_timer()
hashtags_freq = execute_heshtags(lines)
print(Sort(hashtags_freq)[:10])
stop = timeit.default_timer()
print('Time: ', stop - start)
```

```
[('ff', 3581), ('nowplaying', 1809), ('fb', 1402), ('mm', 1029), ('fail', 686),
('random', 622), ('haiti', 591), ('shoutout', 529), ('followfriday', 457), ('music
monday', 452)]
Time:  1.6265050999999175
```

1.1 (Unix)

In [32]:

```
# Extract fist 500,000 lines into "test_set_tweets_500000.txt "
!head -500000 test_set_tweets.txt > test_set_tweets_500000.txt
```

In [73]:

```
# Extract hashtags words and store them into "hashtags_500000.txt"
!grep -P -o "#[^ \t]+" test_set_tweets_500000.txt > hashtags_500000.txt
```

In [74]:

```
# First 10 lines of "hashtags_500000.txt"
!head -10 hashtags_500000.txt
```

```
#confession.
#worstfeeling.:
#FF
#mm.
#niggas.
#dontjudgeme
#nowplaying.
#nowplaying.
#PersonalBelief
#imjustsayin
```

In [75]:

```
# strip out punctuation and convert uppercase to lowercase
!sed 's/#//g' hashtags_500000.txt | sed 's/[^a-zA-Z]//g' | sed -e 's/\(.*\)/\L
\1/' > keywords_500000.txt
```

In [76]:

```
# Calculate frequence of hashtags and store the result into "result_hastags_5000
00.txt"
!sort keywords_500000.txt | uniq --count | sort -nr > result_hastags_500000.txt
```

In [91]:

```
# Result of top 10 hashtags
!head -10 result_hastags_500000.txt
```

```
   3581 ff
   1809 nowplaying
   1402 fb
   1361
   1029 mm
    686 fail
    622 random
    591 haiti
    529 shoutout
    457 followfriday
```

In [92]:

```
# Shell script of 1.1
!cat 1_1.sh
```

```
#!/bin/sh
sed 's/#//g' hashtags_500000.txt | sed 's/[^a-zA-Z]//g' | sed -e
's/\(.*\)/\L\1/' > keywords_500000.txt
sort keywords_500000.txt | uniq --count | sort -nr > result_hastags
_500000.txt
head -10 result_hastags_500000.txt
```

In [101]:

```
# Runtime of 1.1 using Unix command
!time bash 1_1.sh
```

```
   3581 ff
   1809 nowplaying
   1402 fb
   1361
   1029 mm
    686 fail
    622 random
    591 haiti
    529 shoutout
    457 followfriday
0.33user 0.01system 0:00.25elapsed 133%CPU (0avgtext+0avgdata 6436m
axresident)k
0inputs+2232outputs (0major+2558minor)pagefaults 0swaps
```

Discussion:

As is shown above, it takes about **1.63 sec** to find the top 10 hashtags using map reduce approach in Python, while it only takes **0.25 sec** using Unix command. I think this is because Python is an interpreted language and it runs much slower than shell command.

1.2 (1)

In [12]:

```
# Read txt
with open("tweets.txt","r", encoding='utf-8') as file:
    lines = [next(file) for x in range(750000)]
```

In [4]:

```
import re

def extractUsernames(string):
    pattern = re.compile(r"@(\S+)")
    strs = re.findall(pattern, string)

    output = []
    for i in strs:
        output.append(i)
    return output

# Example:
extractUsernames("22077441      10470781081     @Confession.    I can't live with my mama!!! Espe
cially if I don't have my own car!      2010-03-14 09:21:58")
```

Out[4]:

```
['Confession.']
```

In [5]:

```
def mapper_usernames_line(line):
    words = extractUsernames(line)
    output = []
    for word in words:
        if word:
            output.append((word,1))
    return output

# Example:
mapper_usernames_line("22077441 10470781081     @Confession.    I can't live with my mama!!! Espe
cially if I don't have my own car!      2010-03-14 09:21:58")
```

Out[5]:

```
[('Confession.', 1)]
```

In [6]:

```
def mapper_usernames(lines):
    output = []
    for line in lines:
        list = mapper_usernames_line(line)
        if list:
            output += list
    return output

#Example:
test = ["@John. 2010", "@Jerry 2011", "@Tom 2012", "@Jerry 2013"]
mapper_usernames(test)
```

Out[6]:

```
[('John.', 1), ('Jerry', 1), ('Tom', 1), ('Jerry', 1)]
```

In [7]:

```
def combiner_usernames(mapper_output):
    groups = {} # group by key values
    for item in mapper_output:
        k = item[0]
        v = item[1]
        if k not in groups:
            groups[k] = [v]
        else:
            groups[k].append(v)
    return groups

#Example:
combiner_usernames(mapper_usernames(test))
```

Out[7]:

```
{'John.': [1], 'Jerry': [1, 1], 'Tom': [1]}
```

In [8]:

```
def reducer_usernames(keyWord, counts):
    return (keyWord, sum(counts))

reducer_usernames( 'jerry',[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

Out[8]:

```
('jerry', 14)
```

In [14]:

```
def execute_usernames(lines):
    groups = combiner_usernames(mapper_usernames(lines))
    output = [reducer_usernames(k,v) for k,v in groups.items()]
    output.sort()
    return output

usernames_freq = execute_usernames(lines)
```

In [15]:

```
def Sort(orig):

    orig.sort(key = lambda x: x[1], reverse = True)
    return orig

print(Sort(usernames_freq)[:10])
```

```
[('RevRunWisdom:', 1234), ('listensto', 939), ('DonnieWahlberg', 525), ('OGmuscle
s', 441), ('addthis', 429), ('breatheitin', 411), ('justinbieber', 354), ('MAV25',
347), ('karlievoice', 305), ('mtgcolorpie', 291)]
```

In [16]:

```
import timeit

start = timeit.default_timer()
usernames_freq = execute_usernames(lines)
print(Sort(usernames_freq)[:10])
stop = timeit.default_timer()
print('Time: ', stop - start)
```

```
[('RevRunWisdom:', 1234), ('listensto', 939), ('DonnieWahlberg', 525), ('OGmuscle
s', 441), ('addthis', 429), ('breatheitin', 411), ('justinbieber', 354), ('MAV25',
347), ('karlievoice', 305), ('mtgcolorpie', 291)]
Time:  3.0066348999999946
```

1.2.1 (Unix)

In [13]:

```
# Extract fist 250,000 lines into "training_set_tweets_250000.txt"
!head -250000 training_set_tweets.txt > training_set_tweets_250000.txt
```

In [14]:

```
# First 10 lines in "training_set_tweets_250000.txt"
!head -10 training_set_tweets_250000.txt
```

In [15]:

```
# Join the first 500,000 lines from "test_set_tweets_500000.txt" and the first 2
50,000 lines from "training_set
# _tweets_250000.txt" into "tweets.txt"
!cat test_set_tweets_500000.txt training_set_tweets_250000.txt > tweets.txt
```

In [16]:

```
# Number of lines in "tweets.txt"
!wc -l tweets.txt
```

```
750000 tweets.txt
```

In [84]:

```
# Extract username and store them into "tweets_username.txt"
!grep -P -o "@[^ \t]+" tweets.txt > tweets_username.txt
```

In [85]:

```
# First 10 lines in "tweets_username.txt"
!head -10 tweets_username.txt
```

```
@LovelyJ_Janelle
@Iam_MarkyMark
@Iam_MarkyMark
@Iam_MarkyMark
@seanlamar919
@TRenee3
@LovelyJ_Janelle
@seanlamar919
@seanlamar919
@Iam_MarkyMark:
```

In [86]:

```
# Calculate frequence of hashtags and store the result into "result_username_750000.txt"
!sort tweets_username.txt | uniq --count | sort -nr > result_username_750000.txt
```

In [87]:

```
# Result of top 10 usernames
! head -10 result_username_750000.txt
```

```
   1234 @RevRunWisdom:
    939 @listensto
    525 @DonnieWahlberg
    441 @OGmuscles
    419 @addthis
    411 @breatheitin
    354 @justinbieber
    347 @MAV25
    303 @karlievoice
    291 @mtgcolorpie
```

In [102]:

```
# Shell script for 1.2 (1)
!cat 1_2.sh
```

```
#!/bin/sh
grep -P -o "@[^ \t]+" tweets.txt > tweets_username.txt
sort tweets_username.txt | uniq --count | sort -nr > result_usernam
e_750000.txt
head -10 result_username_750000.txt
```

In [103]:

```
# Runtime of 1.2 (1) using Unix command
!time bash 1_2.sh
```

```
   1234 @RevRunWisdom:
    939 @listensto
    525 @DonnieWahlberg
    441 @OGmuscles
    419 @addthis
    411 @breatheitin
    354 @justinbieber
    347 @MAV25
    303 @karlievoice
    291 @mtgcolorpie
2.06user 0.04system 0:01.12elapsed 187%CPU (0avgtext+0avgdata 55920
maxresident)k
0inputs+25128outputs (0major+16292minor)pagefaults 0swaps
```

Discussion:

As is shown above, it takes about **3.01 sec** to find the top 10 hashtags using map reduce approach in Python, while it only takes **1.12 sec** using Unix command.

1.2 (2)

In [27]:

```
import re

def extractTwohashtags(string):
    pattern = re.compile(r"#\S+#\S+")
    strs = re.search(pattern, string)
    output = []
    if strs:
        output.append(strs.group(0))
    else:
        output.append([])
    return output

# Example:
print(extractTwohashtags("22077441	10470781081	#Confession.	I can't live with my mam
a!!! Especially if I don't have my own car!	2010-03-14 09:21:58"))
print(extractTwohashtags("22077441	10470781081	#Confession#Disappointment#Desperation.
I can't live with my mama!!! Especially if I don't have my own car!	2010-03-14 09:21:58"))
```

```
[[]]
['#Confession#Disappointment#Desperation.']
```

In [18]:

```
def mapper_twohashtags_line(line):
    words = extractTwohashtags(line)
    output = []
    for word in words:
        if word:
            output.append((word,1))
    return output

# Example:
mapper_twohashtags_line("22077441	10470781081	#Confession#Disappointment	I can't liv
e with my mama!!! Especially if I don't have my own car!	2010-03-14 09:21:58")
```

Out[18]:

```
[('#Confession#Disappointment', 1)]
```

In [19]:

```
def mapper_twohashtags(lines):
    output = []
    for line in lines:
        list = mapper_twohashtags_line(line)
        if list:
            output += list
    return output

#Example:
test = ["#John.#2010", "#Jerry#2013", "#Tom2012", "#Jerry#2013"]
mapper_twohashtags(test)
```

Out[19]:

```
[('#John.#2010', 1), ('#Jerry#2013', 1), ('#Jerry#2013', 1)]
```

In [20]:

```
def combiner_twohashtags(mapper_output):
    groups = {} # group by key values
    for item in mapper_output:
        k = item[0]
        v = item[1]
        if k not in groups:
            groups[k] = [v]
        else:
            groups[k].append(v)
    return groups

#Example:
combiner_twohashtags(mapper_twohashtags(test))
```

Out[20]:

```
{'#John.#2010': [1], '#Jerry#2013': [1, 1]}
```

In [21]:

```
def reducer_twohashtags(keyWord, counts):
    return (keyWord, sum(counts))

reducer_twohashtags( 'jerry',[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

Out[21]:

```
('jerry', 14)
```

In [22]:

```
def execute_twohashtags(lines):
    groups = combiner_twohashtags(mapper_twohashtags(lines))
    output = [reducer_twohashtags(k,v) for k,v in groups.items()]
    output.sort()
    return output

twohashtags_freq = execute_twohashtags(lines)
```

In [23]:

```
def Sort(orig):

    orig.sort(key = lambda x: x[1], reverse = True)
    return orig

print(Sort(twohashtags_freq)[:10])
```

```
[('#affiliate#marketing', 8), ('####', 5), ('#Celebrity,#Philanthropy', 4), ('#39;
Green&#39;', 3), ('#39;What&#39;s', 3), ('#39;streaming&#39;', 3), ('#??PFoundersd
ay#??PFoundersday', 3), ('#39;A&#39;', 2), ('#39;SNL&#39;:', 2), ('#39;Twilight&#3
9;', 2)]
```

In [24]:

```
import timeit

start = timeit.default_timer()
usernames_freq = execute_twohashtags(lines)
print(Sort(twohashtags_freq)[:10])
stop = timeit.default_timer()
print('Time: ', stop - start)
```

```
[('#affiliate#marketing', 8), ('####', 5), ('#Celebrity,#Philanthropy', 4), ('#39;
Green&#39;', 3), ('#39;What&#39;s', 3), ('#39;streaming&#39;', 3), ('#??PFoundersd
ay#??PFoundersday', 3), ('#39;A&#39;', 2), ('#39;SNL&#39;:', 2), ('#39;Twilight&#3
9;', 2)]
Time:  1.9740761999999847
```

1.2.2 (Unix)

In [71]:

```
# Extract username and store them into "tweets_username.txt"
!grep -P -o "#[^ \t]+#[^ \t]+" tweets.txt > tweets_twohashtags.txt
```

In [88]:

```
# First 10 lines in "tweets_twohashtags.txt"
!head -10 tweets_twohashtags.txt
```

```
#trueshit...#scaryshit...but
#!%#$^!@%&
#honey.......#imjussayn
#pause.....#megapause
#9:#Virtualization
#8:#Offshore
#thatisall--#agree
#FF--#FF
#LENT....#SMH
#UCanTakeTheKidOutHoodBut#UCanTakeTheKidOutHoodBut#UCanTakeTheKidOu
tHoodBut#UCanTakeTheKidOutHoodBut#UCanTakeTheKidOutHoodBut
```

In [89]:

```
# Calculate frequence of hashtags and store the result into "result_twohastags_7
50000.txt"
!sort tweets_twohashtags.txt | uniq --count | sort -nr > result_twohashtags_7500
00.txt
```

In [90]:

```
# Result of top 10 tweets that have at least two hashtags
! head -10 result_twohashtags_750000.txt
```

```
      8 #affiliate#marketing
      5 #zewdy#zewdy
      5 #BGC#BGC
      5 ####
      4 #??PFoundersday#??PFoundersday
      4 #Celebrity,#Philanthropy
      3 #AKA#AKA
      3 #39;What&#39;s
      3 #39;streaming&#39;
      3 #39;Green&#39;
```

In [104]:

```
# Shell script for 1.2 (2)
!cat 1_3.sh
```

```
#!/bin/sh
grep -P -o "#[^ \t]+#[^ \t]+" tweets.txt > tweets_twohashtags.txt
sort tweets_twohashtags.txt | uniq --count | sort -nr > result_twoh
ashtags_750000.txt
head -10 result_twohashtags_750000.txt
```

In [105]:

```
# Runtime of 1.2 (2) using Unix command
!time bash 1_3.sh
```

```
      8 #affiliate#marketing
      5 #zewdy#zewdy
      5 #BGC#BGC
      5 ####
      4 #??PFoundersday#??PFoundersday
      4 #Celebrity,#Philanthropy
      3 #AKA#AKA
      3 #39;What&#39;s
      3 #39;streaming&#39;
      3 #39;Green&#39;
0.17user 0.01system 0:00.18elapsed 100%CPU (0avgtext+0avgdata 3108m
axresident)k
0inputs+48outputs (0major+828minor)pagefaults 0swaps
```

Discussion:

As is shown above, it takes about **1.97 sec** to find the top 10 hashtags using map reduce approach in Python, while it only takes **0.18 sec** using Unix command.

2 Finding Reciprocal Followers

In [30]:

```
import pandas as pd

edges_orig = pd.read_csv("./Twitter-dataset/data/edges.csv")
edges = edges_orig.head(500000)
```

In [33]:

```
edges_test = edges_orig.head(1000)
```

In [34]:

```
def mapper_reciprocal(df):
    return list(map(list, df.values))

# Example:
mapper_reciprocal(edges_test)[:20]
```

Out[34]:

```
[[1, 8762940],
 [1, 8762941],
 [1, 688136],
 [1, 8762942],
 [3, 718952],
 [3, 3109655],
 [3, 562897],
 [3, 6],
 [3, 7],
 [3, 12852],
 [3, 90259],
 [3, 8762941],
 [3, 645510],
 [3, 427258],
 [3, 45567],
 [3, 1374301],
 [3, 38253],
 [3, 79994],
 [3, 16],
 [3, 9]]
```

In [35]:

```
def combiner_reciprocal(mapper_output):
    groups = {} # group by key values
    for item in mapper_output:
        k = item[0]
        v = item[1]
        if k not in groups:
            groups[k] = [v]
        else:
            groups[k].append(v)
    return groups

# Example:
# combiner_reciprocal(mapper_reciprocal(edges_test))
```

In [36]:

```
def reducer_reciprocal(userID, followingID, group):
    if followingID in group:
        if userID in group[followingID]:
            return (userID, followingID)

#Example:
g = {1:[2,3],2:[1],3:[2,4]}
reducer_reciprocal(1, 2, g)
```

Out[36]:

(1, 2)

In [37]:

```
def execute_reciprocal(edges):
    map_reciprocal = mapper_reciprocal(edges)
    groups = combiner_reciprocal(map_reciprocal)
    output = []
    for users in map_reciprocal:
            pair =  reducer_reciprocal(users[0],users[1], groups)
            if pair:
                output.append(pair)
    output.sort()
    return output

output = execute_reciprocal(edges)
```

In [38]:

```
import timeit

start = timeit.default_timer()
execute_reciprocal(edges)
stop = timeit.default_timer()
print('Time: ', stop - start)
```

Time: 1.2204441000000088

In [39]:

```
follower_graph = pd.DataFrame(output, columns =['userID', 'followerID'])
follower_graph.to_csv('follower_graph.csv', index=False)
follower_graph
```

Out[39]:

	userID	followerID
0	3682	5276
1	5276	3682
2	13232	18205
3	13232	63255
4	15574	15926
5	15926	15574
6	18205	13232
7	19628	19821
8	19628	20033
9	19821	19628
10	20033	19628
11	22196	76473
12	23503	41422
13	31866	32002
14	32002	31866
15	32173	32452
16	32452	32173
17	33099	62167
18	33884	34046
19	33884	34101
20	34046	33884
21	34101	33884
22	40704	40997
23	40704	41039
24	40997	40704
25	40997	41039
26	40997	62623
27	40997	201063
28	41039	40704
29	41039	40997
...	...	...
38	65411	65435
39	65435	63255
40	65435	65411
41	65435	93260
42	70696	60887
43	70696	70772

	userID	followerID
44	70772	70696
45	76473	22196
46	78182	78464
47	78464	78182
48	80092	80096
49	80096	80092
50	89222	89350
51	89350	89222
52	93260	65435
53	93260	93427
54	93427	93260
55	100591	100721
56	100721	100591
57	102898	122546
58	122546	102898
59	134409	134410
60	134410	134409
61	135546	135684
62	135684	135546
63	192865	192899
64	192899	192865
65	201063	40997
66	201078	201607
67	201607	201078

68 rows × 2 columns

2 (Unix)

In [ ]:

```
# Extract fist 250,000 lines into "training_set_tweets_250000.txt"
!head -500000  edges.csv > edges_500000.csv
```

In [18]:

```
# Swap order is userID is larger than followerID and store the result into "edge
s_500000_dup.csv"
!awk -F "," '{if($1<$2) printf("%d,%d\n", $1,$2);if($1>$2) printf("%d,%d\n",
 $2,$1)}' edges_500000.csv > edges_500000_dup.csv
```

In [19]:

```
# Find pairs that appear twice (reciprocal follower) and store it into "output.c
sv"
!sort edges_500000_dup.csv | uniq --count --repeated > output.csv
```

In [34]:

```
# Report reciprocal followers
!grep -E -o " [0-9]+,[0-9]+$" output.csv | awk -F "," '{printf("%d,%d\n%d,%d\n",
$1,$2, $2,$1)}' > result_reciprocalFollowers.txt
```

```
100591,100721
100721,100591
102898,122546
122546,102898
13232,18205
18205,13232
13232,63255
63255,13232
134409,134410
134410,134409
135546,135684
135684,135546
15574,15926
15926,15574
192865,192899
192899,192865
19628,19821
19821,19628
19628,20033
20033,19628
201063,40997
40997,201063
201078,201607
201607,201078
22196,76473
76473,22196
23503,41422
41422,23503
31866,32002
32002,31866
32173,32452
32452,32173
33099,62167
62167,33099
33884,34046
34046,33884
33884,34101
34101,33884
3682,5276
5276,3682
40704,40997
40997,40704
40704,41039
41039,40704
40997,41039
41039,40997
40997,62623
62623,40997
58783,58875
58875,58783
60887,70696
70696,60887
63255,65435
65435,63255
65411,65435
65435,65411
65435,93260
93260,65435
70696,70772
70772,70696
78182,78464
```

```
78464,78182
80092,80096
80096,80092
89222,89350
89350,89222
93260,93427
93427,93260
```

In [106]:

```
# Number of reciprocal followers: 34 * 2
!grep -E -o " [0-9]+,[0-9]+$" output.csv | awk -F "," '{printf("%d,%d\n%d,%d\n",
$1,$2, $2,$1)}' | wc -l
```

```
68
```

In [115]:

```
# Shell script for 2
!cat 2.sh
```

```
#!/bin/sh
awk -F "," '{if($1<$2) printf("%d,%d\n", $1,$2);if($1>$2) printf("%
d,%d\n", $2,$1)}' edges_500000.csv > edges_500000_dup.csv
sort edges_500000_dup.csv | uniq --count --repeated > output.csv
grep -E -o " [0-9]+,[0-9]+$" output.csv | awk -F "," '{printf("%d,%
d\t%d,%d\t", $1,$2, $2,$1)}'
echo "\n"
```

In [116]:

```
# Runtime of 2 using Unix command
!time bash 2.sh
```

```
100591,100721   100721,100591   102898,122546   122546,102898   132
32,18205        18205,13232     13232,63255     63255,13232     134
409,134410      134410,134409   135546,135684   135684,135546   155
74,15926        15926,15574     192865,192899   192899,192865   196
28,19821        19821,19628     19628,20033     20033,19628     201
063,40997       40997,201063    201078,201607   201607,201078   221
96,76473        76473,22196     23503,41422     41422,23503     318
66,32002        32002,31866     32173,32452     32452,32173     330
99,62167        62167,33099     33884,34046     34046,33884     338
84,34101        34101,33884     3682,5276       5276,3682       407
04,40997        40997,40704     40704,41039     41039,40704     409
97,41039        41039,40997     40997,62623     62623,40997     587
83,58875        58875,58783     60887,70696     70696,60887     632
55,65435        65435,63255     65411,65435     65435,65411     654
35,93260        93260,65435     70696,70772     70772,70696     781
82,78464        78464,78182     80092,80096     80096,80092     892
22,89350        89350,89222     93260,93427     93427,93260     \n
1.53user 0.03system 0:00.71elapsed 219%CPU (0avgtext+0avgdata 55792
maxresident)k
0inputs+13344outputs (0major+14257minor)pagefaults 0swaps
```

Discussion:

As is shown above, it takes about **1.22 sec** to find the top 10 hashtags using map reduce approach in Python, while it only takes **0.71 sec** using Unix command. In this question, I solved it in different ways when using map reduce approach and Unix command. As for map reduce, I used the same algorithm as question 1. However, for Unix command, I used "awk" command to sort each line into ascending order at the begining. By doing so, if two users is pair of reciprocal follower, their ids will appear twice in the output file. Finally, by using "sort" command, we can easily find out pairs that are reciprocal followers. I think this method is faster than map reduce approach.

3 Finding Friends of Friends

In [111]:

```
# Read csv file
import pandas as pd

edges_orig = pd.read_csv("./Twitter-dataset/data/edges.csv")
follower_graph = pd.read_csv('follower_graph.csv')
edges = edges_orig.head(500000)
edges_test = edges_orig.head(5000)
```

In [139]:

```
pairs = list(map(list, follower_graph.values))
```

In [112]:

```
groups_edges = combiner_reciprocal(mapper_reciprocal(edges))
```

In [126]:

```
def mapper_findFriends(userID, group):
    if userID in group:
        return group[userID]

# Example;
mapper_findFriends(1, groups_edges)
```

Out[126]:

```
[8762940, 8762941, 688136, 8762942]
```

In [117]:

```
def mapper_commonFriends(list1, list2):
    return list(set(list1).intersection(list2))

# Example;
mapper_commonFriends([1,2,3,4,5],[2,4])
```

Out[117]:

```
[2, 4]
```

In [140]:

```
def reducer_numOfFriends(list1, list2):
    common = list(set(list1).intersection(list2))
    return len(common)

# Example;
reducer_numOfFriends([1,2,3,4,5],[2,4])
```

Out[140]:

2

In [147]:

```
def execute_commonFriends(edgesGraph, followerGraph, groups):
    output = []
    for pair in followerGraph:
#         print(pair)
        userID = pair[0]
        follerID = pair[1]
        friendOfUser = mapper_findFriends(userID, groups)
        friendOfFoller = mapper_findFriends(follerID, groups)
        output.append((pair, reducer_numOfFriends(friendOfUser,friendOfFoller)))
    return output

output = execute_commonFriends(edges, pairs, groups_edges)
output = sorted(output, key = lambda x: x[1], reverse = True)
output[:20]
```

Out[147]:

```
[([3682, 5276], 714),
 ([5276, 3682], 714),
 ([40704, 40997], 402),
 ([40997, 40704], 402),
 ([40997, 41039], 360),
 ([41039, 40997], 360),
 ([23503, 41422], 352),
 ([41422, 23503], 352),
 ([60887, 70696], 332),
 ([70696, 60887], 332),
 ([135546, 135684], 282),
 ([135684, 135546], 282),
 ([70696, 70772], 259),
 ([70772, 70696], 259),
 ([40704, 41039], 252),
 ([41039, 40704], 252),
 ([13232, 63255], 236),
 ([63255, 13232], 236),
 ([32173, 32452], 194),
 ([32452, 32173], 194)]
```