 November 20, 2015



**LE Minh-Quoc**

DATA MINING

TWITTER PROJECT

# Introduction

In this project, we were given a couple folders of data from Twitter. The data are the tweet that were collected based on an event or a location.

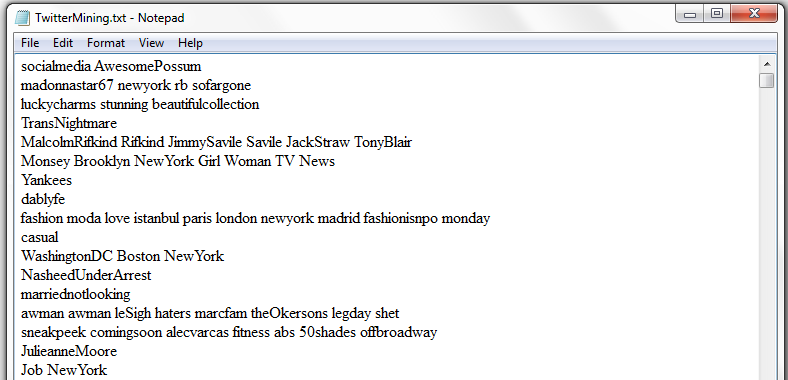
Our mission is to process those data, refine them, analysis and find out the trend that is hidden inside of those data.

The main tasks would follow the following steps:

1. Read data from file, refine and filter data.
2. Build an undirected graph from that refined data.
3. Find the dense subgraph(s)
4. Analysis the results.

# Filtering

This filtering step is the very first task that we have to do in the order to filter out the spamming tweets. The idea behind is that the spamming post (from spammers or robots) will try to replicate the post from other people to get the attention. Hence, they are likely to create the tweet with the hashtags are identical to others. Following the instruction of TA, we will filter out all the tweets which have the identical (in number and in word) hashtags. After that, we store the processed data to an output file that look like the below screen-captured photo:

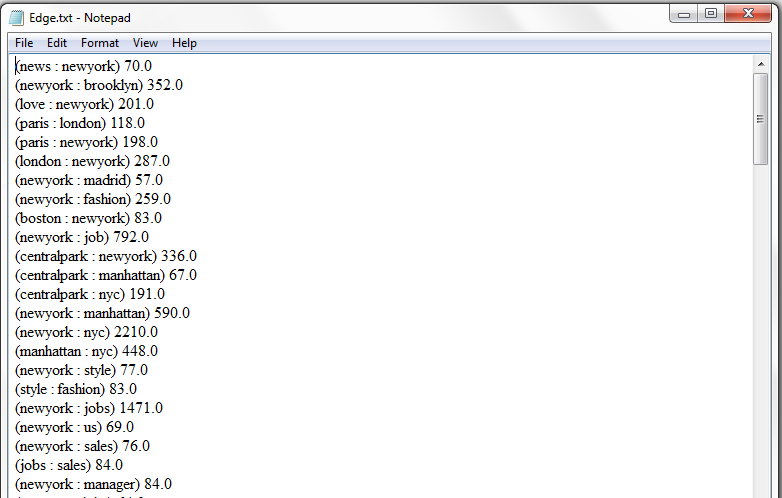


Moreover, we also store the Users’ id into another file. This user txt file will be used after when we need to count the distinct number of users that contribute into the dense subgraph.

# Build the graph

Using the JGraphT library, we were able to build the whole graph G for the data we had. This is a weighted graph that has weights on each of its edges. Hence, we have to calculate the weight for each edge from the data. For each pair of hashtag in the same tweet, we will count 1 for the weight of the edge created by those 2 hashtags.

Finally we will get a data structure in which we have the edge and the its weight.



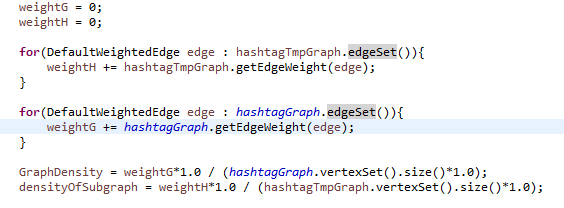
In order for an easier calculation, we calculate the weight of the node (vertex) in graph by the sum of the weight of all edge from/to it (this is an undirected graph, so we do not care “from” or “to”). And store the result into a hashmap, where the data is stored in the form of a pair key-value <K, V>.

# Find the dense subgraph(s)

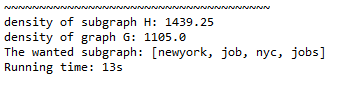
We decided to implement the finding function in 2 ways.

* One is to follow the greedy methods which is to delete from the smallest weight node to find the last nodes that create the dense subgraph. For this method, we will apply the idea of parallel algorithm as in the slide of the Professor.
* The second one is to follow an algorithm that we’ve found on the internet to run thru the big graph, break it into smaller graphs and calculate the density of the smaller graphs. Continue that until we can have many subgraphs with the desired number of nodes. For this method, we can get as many subgraphs as we want (Eg: 20 subgraphs who have maximum 10 vertices).

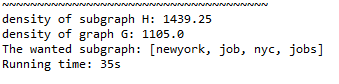
For whatever method we go with, we have to calculate the density of the big graph and the subgraph. We calculate by sum up the weight of all edges and divide by the number of nodes (this is a variant version of the density that we learned in class Number of edge/Number of vertex)



For the first method (parallel greedy), we have the running is quite fast (~14 seconds for the 1st folder (NewYorkOneWeek)). And we got the result as below:



The running time you see above is 13 seconds (exclude the time for reading and filtering data from file). If counted in, it should be 35 seconds:

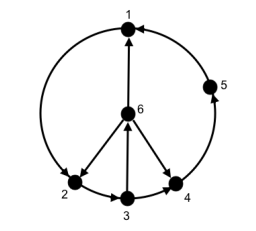


**HOMEWORKS**

**LE MINH QUOC**

# Homework 1:

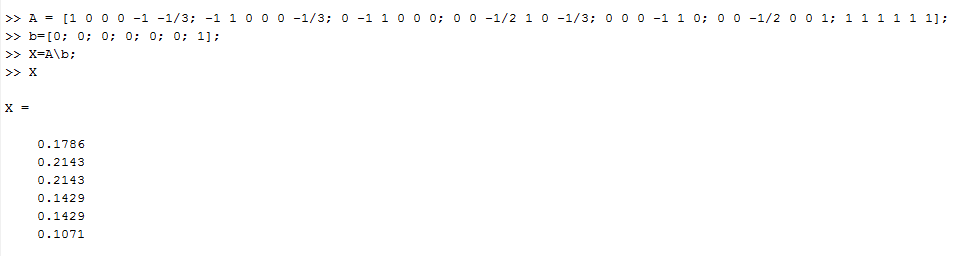
### Exercise 1:



Calling each vertex in the graph G is v1 -> v6. Based on the graph we will have the 6 equations for each of the vertex, as below:

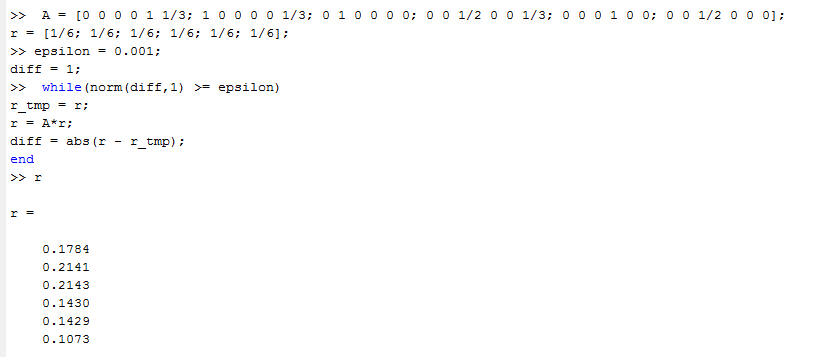
The last equation is come from the attribute of the distributed-weight in the pagerank scores algorithm. From the above multi-equation, we have:

Instead of running this equations by hand, to save time, I ran it in the Matlab and received the result below:



The result is corresponding to v1 -> v6.

Now, we run the Pagerank Algorithm, also in Matlab and obtain the result below:



Obviously, we can see that the result is reasonably the same for the hand-run calculation and the Pagerank algorithm.

### Exercise 2:

## Homework 2:

### Exercise 1: