Lab 4

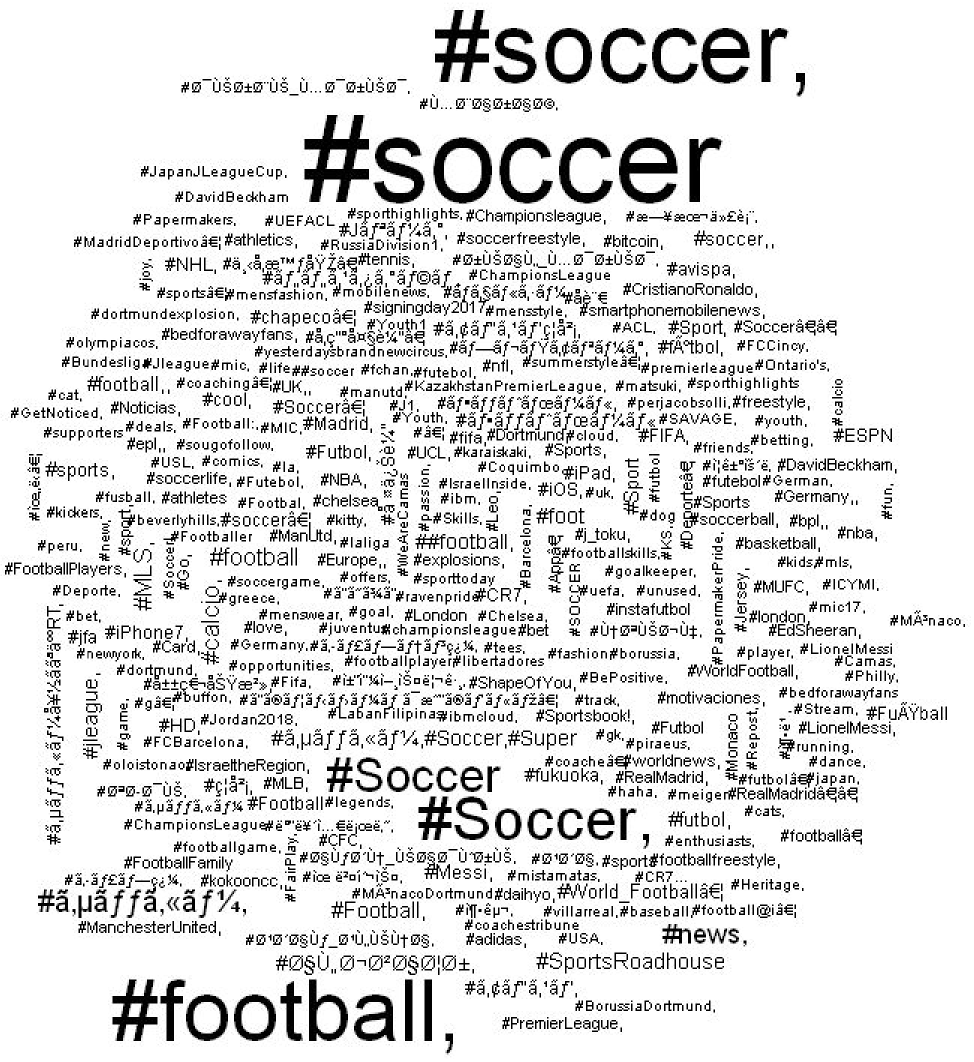
Activity 1: Word Count on tweets

Java file: WordCount.java
JAR file: wordcount.jar
Input to Hadoop: DIC4_1.txt
Output from Hadoop: mapOutput.txt

Steps Performed:

1. Collected tweets using twitteR library in R.
2. Extracted list of #(hashtags) from the collected tweets in a file called 'DIC4_1.txt'.
3. Gave this text file (DIC4_1.txt) as an input to Hadoop.
4. The output received was count for every hashtag. The output file was 'mapOutput.txt'.
5. This output file (mapOutput.txt) was used to create the word-cloud using the 'wordcloud' library.

WordCloud


#soccer,
#soccer
#Soccer
#Soccer,
#football,
#football
#soccer,,
#news,
#SportsRoadhouse
#Football,
#Messi,
#footballl,,
#ESPN
#FIFA,
#sports,
#jleague
#MLS
#Calcio

Activity 2: Word co-occurrence on tweets

Steps Performed:

1. Collected tweets using twitteR library in R.
2. Extracted the text field from each tweet.
3. Performed pre-processing (removed punctuation marks, digits,emoticons etc) so that hadoop can get a valid input in the form of text. Stored this pre-processed output in Activity2.txt file.

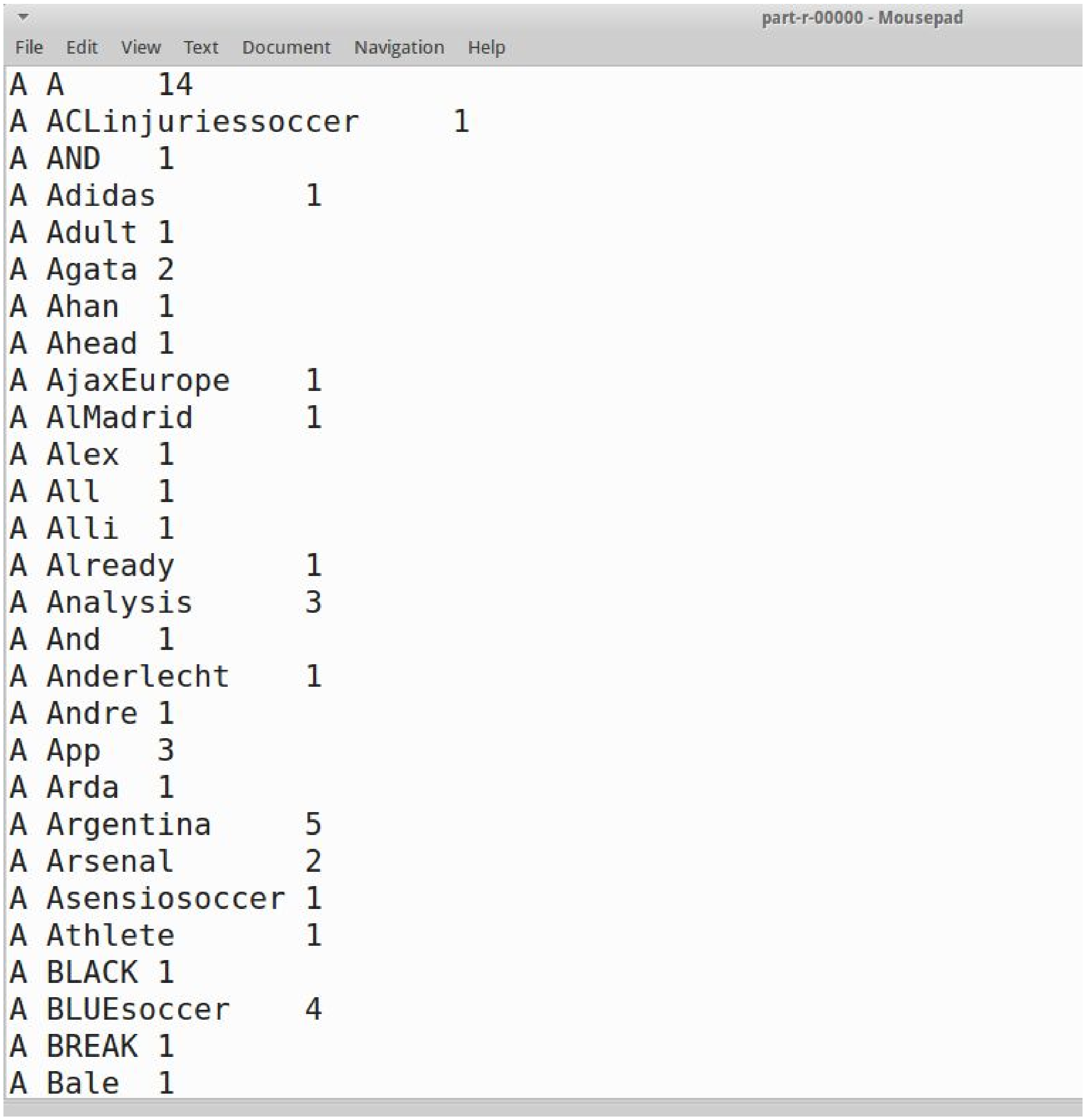
Pairs:

Java file: Pairs.java
JAR file: pairs.jar
Input to Hadoop: Activity2.txt
*Output from Hadoop: Check folder '*Activity2_Pairs_output*'*

Steps Performed:

1. The input used was Activity2.txt.
2. Assumption: All words appearing in a sentence are considered to be co-occurring with each other.

Output



```
A A	14
A ACLinjuriessoccer	1
A AND	1
A Adidas	1
A Adult	1
A Agata	2
A Ahan	1
A Ahead	1
A AjaxEurope	1
A AlMadrid	1
A Alex	1
A All	1
A Alli	1
A Already	1
A Analysis	3
A And	1
A Anderlecht	1
A Andre	1
A App	3
A Arda	1
A Argentina	5
A Arsenal	2
A Asensiosoccer	1
A Athlete	1
A BLACK	1
A BLUEsoccer	4
A BREAK	1
A Bale	1
```

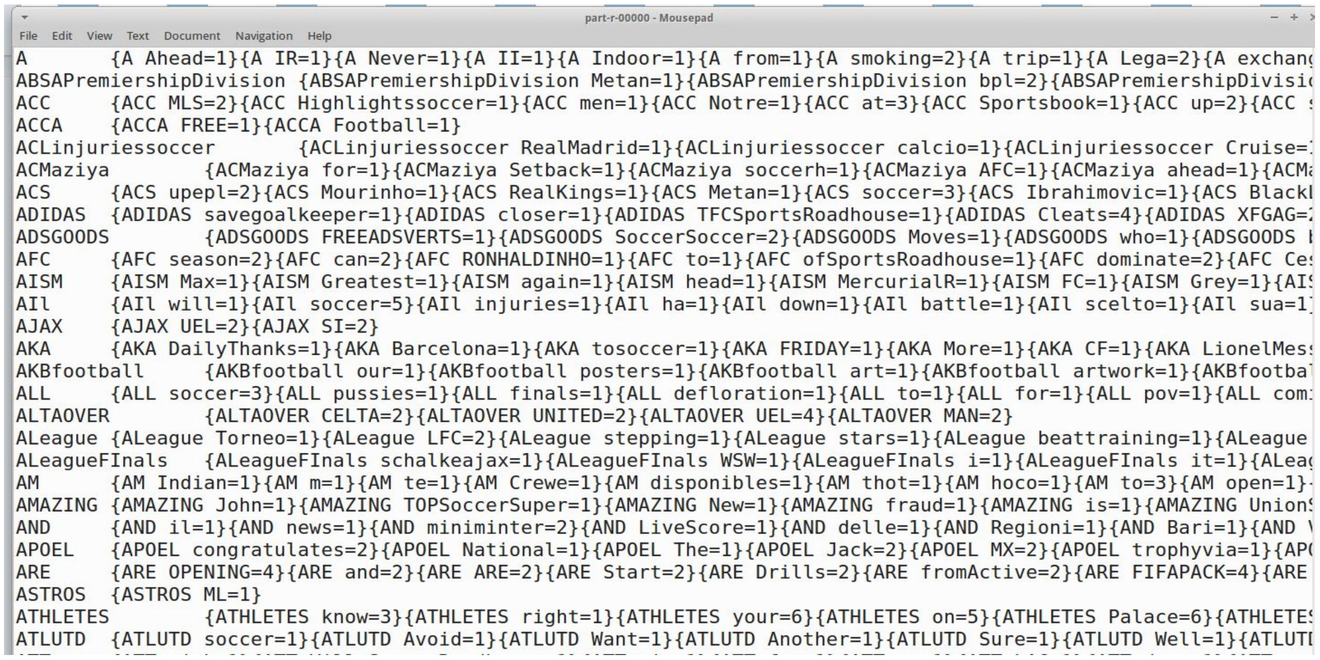
Stripes:

Java file: Stripes.java
JAR file: stripes.jar
Input to Hadoop: Activity2.txt
*Output from Hadoop: Check folder '*Activity2_Stripes_output*'*

Steps Performed:

1. The input used was Activity2.txt.
2. Assumption: All words appearing in a sentence are considered to be co-occurring with each other.
3. In Mapper for every word, I considered all the words in that sentence as it's neighbour and emitted a key-value pair<word neighbor,1>.
4. In Reducer for every key I maintain a hashmap(PrintMapWritable tempMap) which initially has nothing. Each time I come across a new key, I put it in that map. For every repeating key in the map, I increment the count by 1. Finally I emit the <key,tempMap> i.e for every key I emit key and its map.

Output



```
A       {A Ahead=1}{A IR=1}{A Never=1}{A II=1}{A Indoor=1}{A from=1}{A smoking=2}{A trip=1}{A Lega=2}{A exchang
ABSAPremiershipDivision {ABSAPremiershipDivision Metan=1}{ABSAPremiershipDivision bpl=2}{ABSAPremiershipDivisi
ACC     {ACC MLS=2}{ACC Highlightssoccer=1}{ACC men=1}{ACC Notre=1}{ACC at=3}{ACC Sportsbook=1}{ACC up=2}{ACC
ACCA    {ACCA FREE=1}{ACCA Football=1}
ACLinjuriessoccer       {ACLinjuriessoccer RealMadrid=1}{ACLinjuriessoccer calcio=1}{ACLinjuriessoccer Cruise=
ACMaziya        {ACMaziya for=1}{ACMaziya Setback=1}{ACMaziya soccerh=1}{ACMaziya AFC=1}{ACMaziya ahead=1}{ACMa
ACS     {ACS upepl=2}{ACS Mourinho=1}{ACS RealKings=1}{ACS Metan=1}{ACS soccer=3}{ACS Ibrahimovic=1}{ACS Black
ADIDAS  {ADIDAS savegoalkeeper=1}{ADIDAS closer=1}{ADIDAS TFCSportsRoadhouse=1}{ADIDAS Cleats=4}{ADIDAS XFGAG=
ADSGOODS        {ADSGOODS FREEADSVERTS=1}{ADSGOODS SoccerSoccer=2}{ADSGOODS Moves=1}{ADSGOODS who=1}{ADSGOODS
AFC     {AFC season=2}{AFC can=2}{AFC RONHALDINHO=1}{AFC to=1}{AFC ofSportsRoadhouse=1}{AFC dominate=2}{AFC Ce
AISM    {AISM Max=1}{AISM Greatest=1}{AISM again=1}{AISM head=1}{AISM MercurialR=1}{AISM FC=1}{AISM Grey=1}{AI
AIl     {AIl will=1}{AIl soccer=5}{AIl injuries=1}{AIl ha=1}{AIl down=1}{AIl battle=1}{AIl scelto=1}{AIl sua=1
AJAX    {AJAX UEL=2}{AJAX SI=2}
AKA     {AKA DailyThanks=1}{AKA Barcelona=1}{AKA tosoccer=1}{AKA FRIDAY=1}{AKA More=1}{AKA CF=1}{AKA LionelMes
AKBfootball     {AKBfootball our=1}{AKBfootball posters=1}{AKBfootball art=1}{AKBfootball artwork=1}{AKBfootba
ALL     {ALL soccer=3}{ALL pussies=1}{ALL finals=1}{ALL defloration=1}{ALL to=1}{ALL for=1}{ALL pov=1}{ALL com
ALTAOVER        {ALTAOVER CELTA=2}{ALTAOVER UNITED=2}{ALTAOVER UEL=4}{ALTAOVER MAN=2}
ALeague {ALeague Torneo=1}{ALeague LFC=2}{ALeague stepping=1}{ALeague stars=1}{ALeague beattraining=1}{ALeague
ALeagueFInals   {ALeagueFInals schalkeajax=1}{ALeagueFInals WSW=1}{ALeagueFInals i=1}{ALeagueFInals it=1}{ALea
AM      {AM Indian=1}{AM m=1}{AM te=1}{AM Crewe=1}{AM disponibles=1}{AM thot=1}{AM hoco=1}{AM to=3}{AM open=1}
AMAZING {AMAZING John=1}{AMAZING TOPSoccerSuper=1}{AMAZING New=1}{AMAZING fraud=1}{AMAZING is=1}{AMAZING Union
AND     {AND il=1}{AND news=1}{AND miniminter=2}{AND LiveScore=1}{AND delle=1}{AND Regioni=1}{AND Bari=1}{AND
APOEL   {APOEL congratulates=2}{APOEL National=1}{APOEL The=1}{APOEL Jack=2}{APOEL MX=2}{APOEL trophyvia=1}{AP
ARE     {ARE OPENING=4}{ARE and=2}{ARE ARE=2}{ARE Start=2}{ARE Drills=2}{ARE fromActive=2}{ARE FIFAPACK=4}{ARE
ASTROS  {ASTROS ML=1}
ATHLETES        {ATHLETES know=3}{ATHLETES right=1}{ATHLETES your=6}{ATHLETES on=5}{ATHLETES Palace=6}{ATHLETE
ATLUTD  {ATLUTD soccer=1}{ATLUTD Avoid=1}{ATLUTD Want=1}{ATLUTD Another=1}{ATLUTD Sure=1}{ATLUTD Well=1}{ATLUT
```

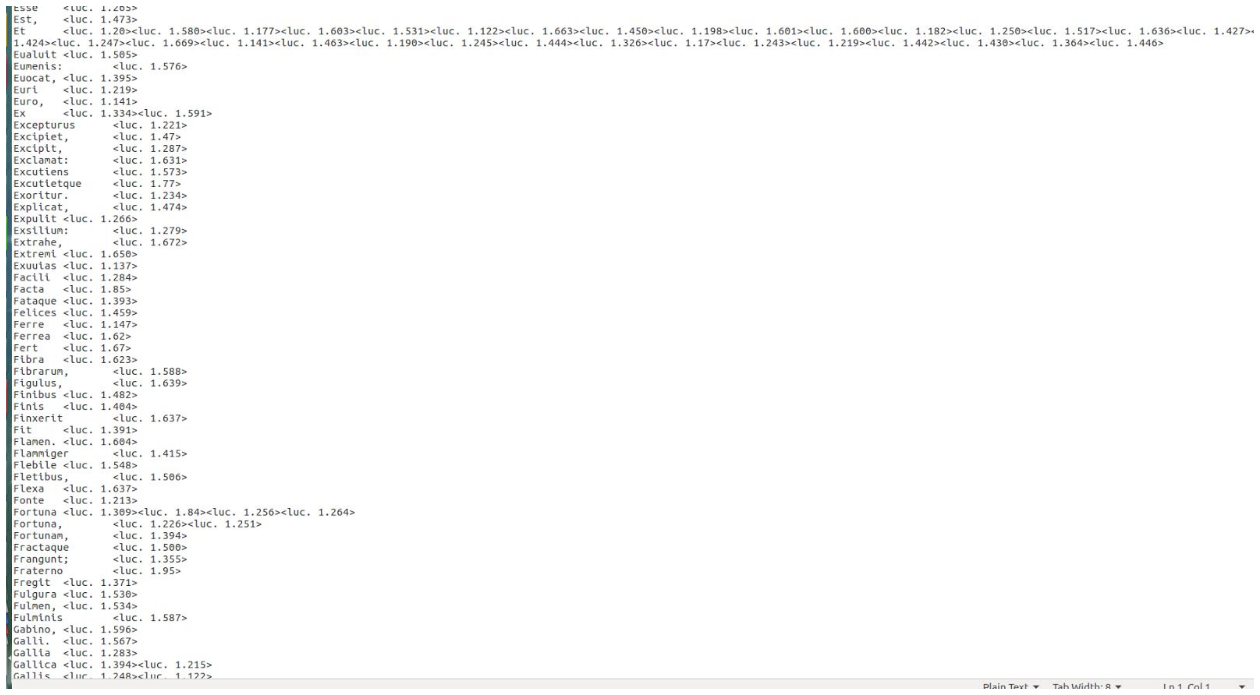
Featured Activity 1: Word Count on Classical Latin text

Java file: FeaturedActivity1.java
JAR file: featuredactivity1.jar
Input to Hadoop: Check folder 'FeaturedActivity1' in 'Inputs' folder
*Output from Hadoop: Check folder '*FeaturedActivity1_output*'*

Steps Performed:

1. Performed the steps mentioned in the lab4 algorithm.
2. Used new_lemmatizer.csv to perform lemmatization (obtain lemmas).
3. In Constructor, created a hashmap 'x' that stores all the words as keys and the lemmas as values. The values column will contain all lemmas separated by space(" ").
4. In Mapper class, for each line of a document I am splitting the line on ">" and then separating location and the content of the line.
5. For every word in the content I am normalizing it and emitting the word and its location.
6. In Reducer class for every key(word) that I receive, I check whether it's in the 'x'. If yes then i find out all the lemmas for that word and for each lemma, I find out all the locations and concatenate them in a single string and emit the lemma and this string. If no then I emit the word and all its corresponding locations where it was found

Output



```
Esse    <luc. 1.265>
Est,    <luc. 1.473>
Et      <luc. 1.20><luc. 1.580><luc. 1.177><luc. 1.603><luc. 1.531><luc. 1.122><luc. 1.663><luc. 1.450><luc. 1.198><luc. 1.601><luc. 1.600><luc. 1.182><luc. 1.250><luc. 1.517><luc. 1.636><luc. 1.427>
1.424><luc. 1.247><luc. 1.669><luc. 1.141><luc. 1.463><luc. 1.190><luc. 1.245><luc. 1.444><luc. 1.326><luc. 1.17><luc. 1.243><luc. 1.219><luc. 1.442><luc. 1.430><luc. 1.364><luc. 1.446>
Eualuit <luc. 1.505>
Eumenis:        <luc. 1.576>
Euocat, <luc. 1.395>
Euri    <luc. 1.219>
Euro,   <luc. 1.141>
Ex      <luc. 1.334><luc. 1.591>
Excepturus      <luc. 1.221>
Excipiet,       <luc. 1.47>
Excipit,        <luc. 1.287>
Exclamat:       <luc. 1.631>
Excutiens       <luc. 1.573>
Excutietque     <luc. 1.77>
Exoritur.       <luc. 1.234>
Explicat,       <luc. 1.474>
Expulit <luc. 1.266>
Exsilium:       <luc. 1.279>
Extrahe,        <luc. 1.672>
Extremi <luc. 1.650>
Exuuias <luc. 1.137>
Facili  <luc. 1.284>
Facta   <luc. 1.85>
Fataque <luc. 1.393>
Felices <luc. 1.459>
Ferre   <luc. 1.147>
Ferrea  <luc. 1.62>
Fert    <luc. 1.67>
Fibra   <luc. 1.623>
Fibrarum,       <luc. 1.588>
Figulus,        <luc. 1.639>
Finibus <luc. 1.482>
Finis   <luc. 1.404>
Finxerit        <luc. 1.637>
Fit     <luc. 1.391>
Flamen. <luc. 1.604>
Flammiger       <luc. 1.415>
Flebile <luc. 1.548>
Fletibus,       <luc. 1.506>
Flexa   <luc. 1.637>
Fonte   <luc. 1.213>
Fortuna <luc. 1.309><luc. 1.84><luc. 1.256><luc. 1.264>
Fortuna,        <luc. 1.226><luc. 1.251>
Fortunam,       <luc. 1.394>
Fractaque       <luc. 1.500>
Frangunt;       <luc. 1.355>
Fraterno        <luc. 1.95>
Fregit  <luc. 1.371>
Fulgura <luc. 1.530>
Fulmen, <luc. 1.534>
Fulminis        <luc. 1.587>
Gabino, <luc. 1.596>
Galli.  <luc. 1.567>
Gallia  <luc. 1.283>
Gallica <luc. 1.394><luc. 1.215>
Gallis  <luc. 1.248><luc. 1.122>
```

Featured Activity 2: Word co-occurrence on Classical Latin text

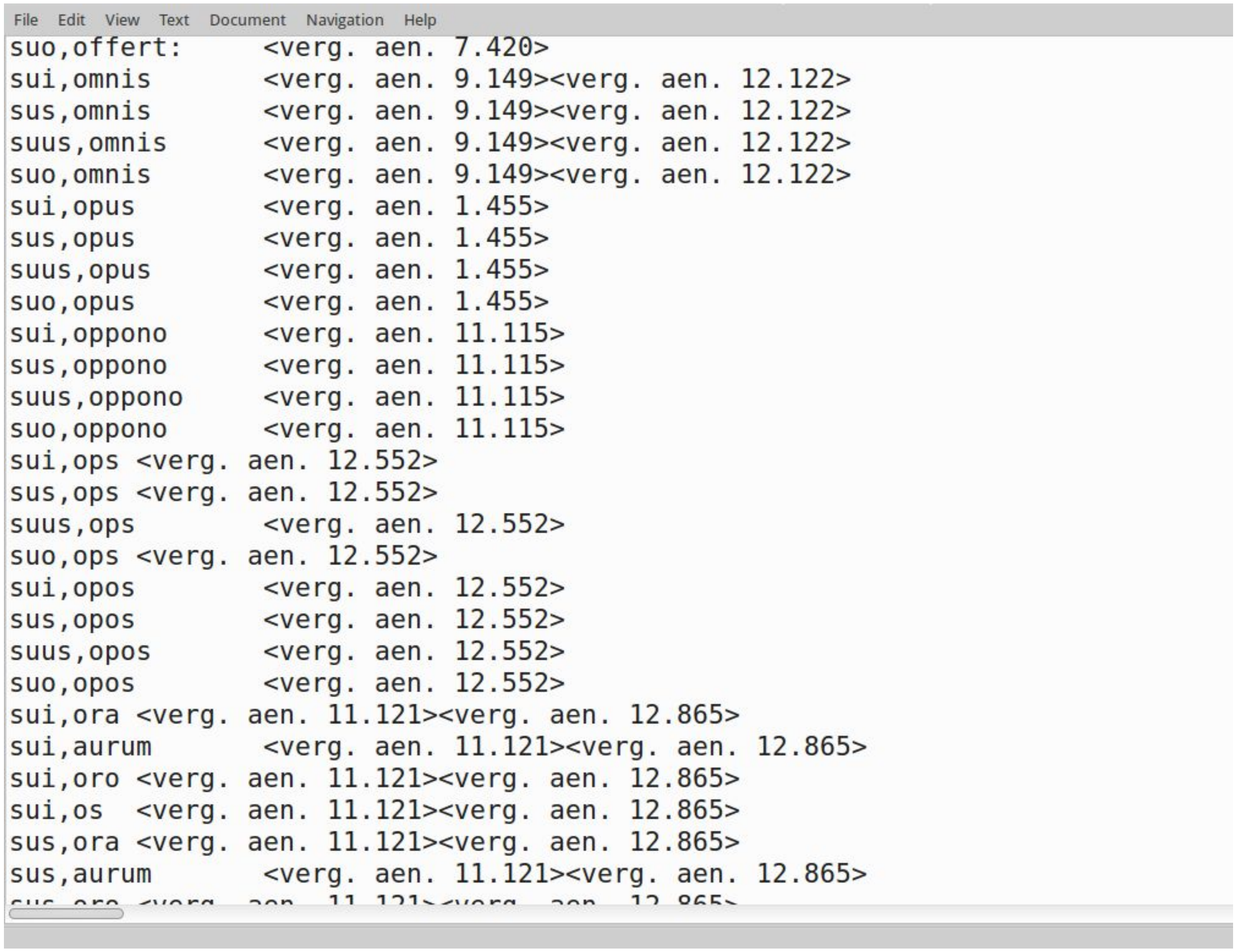
Part A:

Java file: LatinCoccurence1.java
JAR file: latincoccurence1.jar
Input to Hadoop: All the folders inside 'FeaturedActivity2' folder which is present in 'Inputs' folder
*Output from Hadoop: Check folder '*FeaturedActivity2a_output*'*

Steps Performed:

1. Used new_lemmatizer.csv to perform lemmatization (obtain lemmas).
2. In Constructor, created a hashmap 'x' that stores all the words as keys and the lemmas as values. The values column will contain all lemmas separated by space(" ").
3. In Mapper class, for each line of a document I am splitting the line on ">" and then separating location and the content of the line.
4. After that I normalize the word, and for each word I find a neighbour, normalize it, and emit the co-occurring words and the word location as <"word,neighbor",location>.
5. In reducer I get the key as(word:neighbor). I split and check if the word is present in 'x'. Also I check if the neighbor is present in 'x'. If yes then for each combination of word's lemma and neighbor's lemma I emit the key value pair as<"word's lemma,neighbor's lemma",location>. If no then I emit word and neighbor as it is <"word,neighbor",location>.

Output



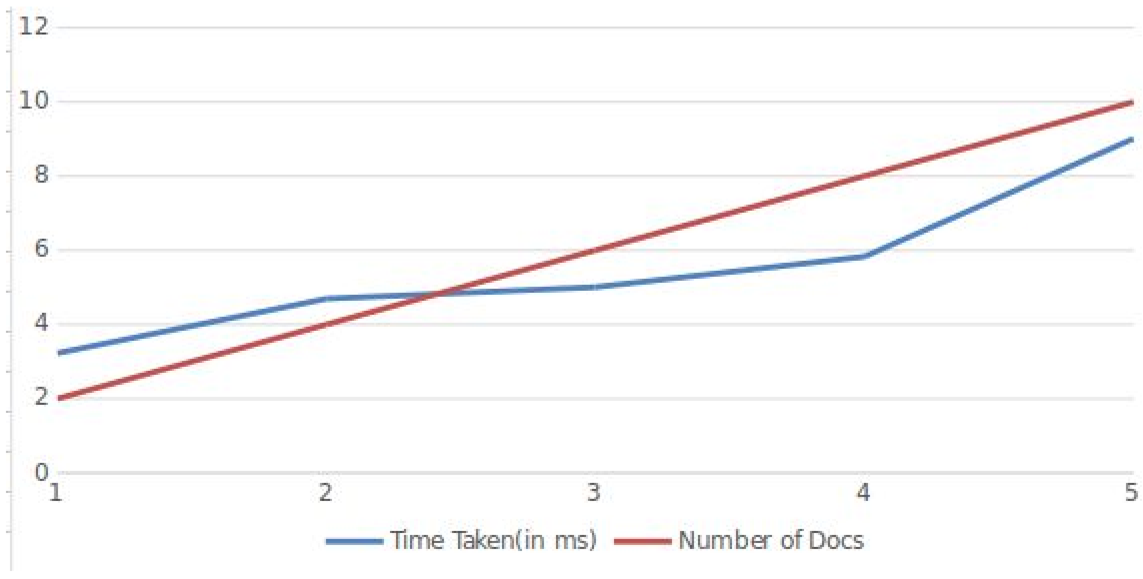
```
suo,offert:     <verg. aen. 7.420>
sui,omnis       <verg. aen. 9.149><verg. aen. 12.122>
sus,omnis       <verg. aen. 9.149><verg. aen. 12.122>
suus,omnis      <verg. aen. 9.149><verg. aen. 12.122>
suo,omnis       <verg. aen. 9.149><verg. aen. 12.122>
sui,opus        <verg. aen. 1.455>
sus,opus        <verg. aen. 1.455>
suus,opus       <verg. aen. 1.455>
suo,opus        <verg. aen. 1.455>
sui,oppono      <verg. aen. 11.115>
sus,oppono      <verg. aen. 11.115>
suus,oppono     <verg. aen. 11.115>
suo,oppono      <verg. aen. 11.115>
sui,ops <verg. aen. 12.552>
sus,ops <verg. aen. 12.552>
suus,ops        <verg. aen. 12.552>
suo,ops <verg. aen. 12.552>
sui,opos        <verg. aen. 12.552>
sus,opos        <verg. aen. 12.552>
suus,opos       <verg. aen. 12.552>
suo,opos        <verg. aen. 12.552>
sui,ora <verg. aen. 11.121><verg. aen. 12.865>
sui,aurum       <verg. aen. 11.121><verg. aen. 12.865>
sui,oro <verg. aen. 11.121><verg. aen. 12.865>
sui,os  <verg. aen. 11.121><verg. aen. 12.865>
sus,ora <verg. aen. 11.121><verg. aen. 12.865>
sus,aurum       <verg. aen. 11.121><verg. aen. 12.865>
```

Plot:

Reference: Book 1.xlsx

Steps Performed:

1. I took a number of readings and recorded the time taken by each reading. I set the number of documents as 2,4,6,8 and 10 recorded the time taken during each run. I did this by adding time before my execution command. Ex: time hadoop jar fa1.jar FeaturedActivity1 ~/LatinInputText/Downloads/2latindocs ~/fa1
2. I drew a plot of time vs number of documents processed. (For reference use Book 1.xlsx)



Observations:

1. We can see that as the number of documents to be processed by Hadoop increases, its execution time also increases.

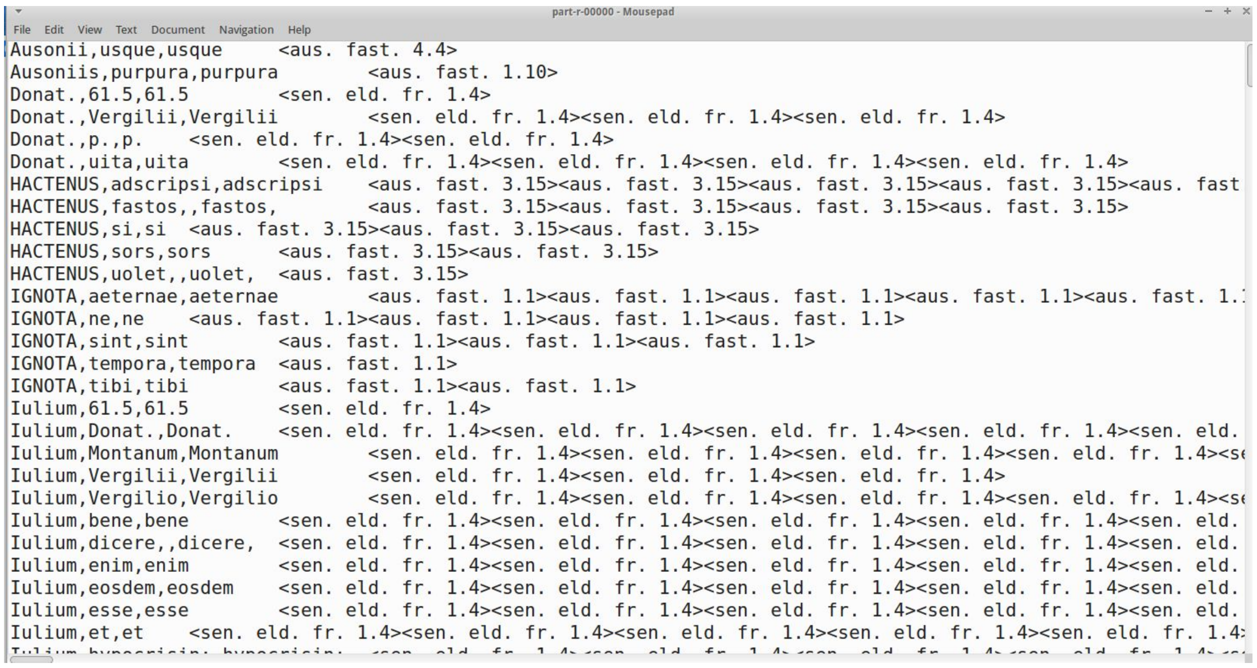
Part B:

Java file: LatinCoccurence2.java
JAR file: latincoccurence2.jar
Input to Hadoop: All the folders inside ‘FeaturedActivity2’ folder which is present in ‘Inputs’ folder
*Output from Hadoop: Check folder ‘*FeaturedActivity2b_output*’*

Steps Performed:

1. The same as in part A except here I am adding another loop for the 2nd neighbor and emitting key value pair as <”word, neighbor1, neighbor2”,location> every time in mapper as well as reducer.

Output



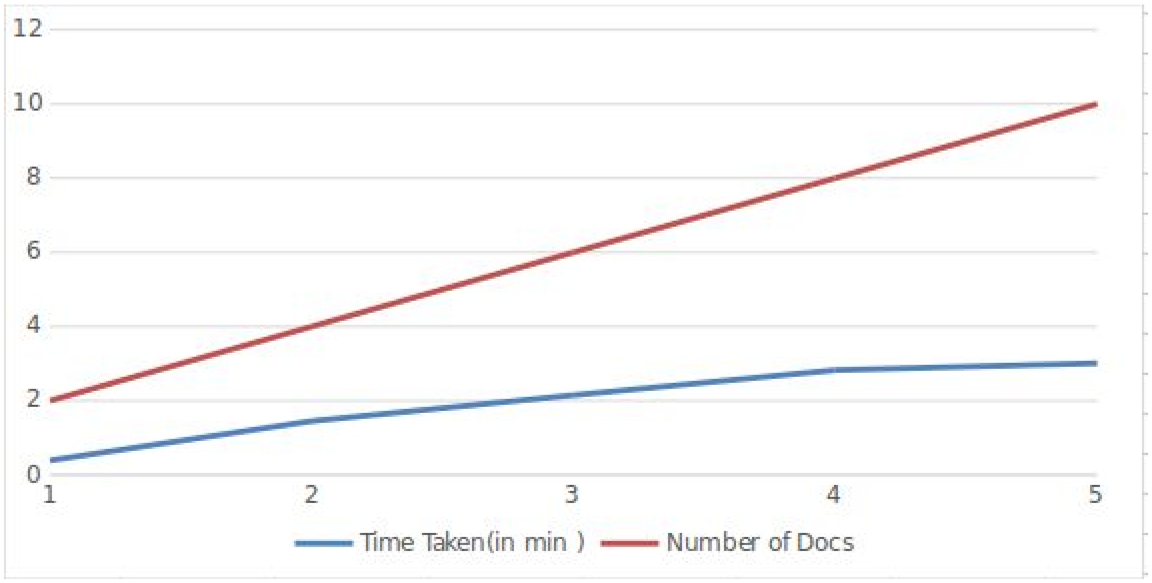
```
Ausonii,usque,usque	<aus. fast. 4.4>
Ausoniis,purpura,purpura	<aus. fast. 1.10>
Donat.,61.5,61.5	<sen. eld. fr. 1.4>
Donat.,Vergilii,Vergilii	<sen. eld. fr. 1.4><sen. eld. fr. 1.4><sen. eld. fr. 1.4>
Donat.,p.,p.	<sen. eld. fr. 1.4><sen. eld. fr. 1.4>
Donat.,uita,uita	<sen. eld. fr. 1.4><sen. eld. fr. 1.4><sen. eld. fr. 1.4><sen. eld. fr. 1.4>
HACTENUS,adscripsi,adscripsi	<aus. fast. 3.15><aus. fast. 3.15><aus. fast. 3.15><aus. fast. 3.15><aus. fast
HACTENUS,fastos,,fastos,	<aus. fast. 3.15><aus. fast. 3.15><aus. fast. 3.15><aus. fast. 3.15>
HACTENUS,si,si	<aus. fast. 3.15><aus. fast. 3.15><aus. fast. 3.15>
HACTENUS,sors,sors	<aus. fast. 3.15><aus. fast. 3.15>
HACTENUS,uolet,,uolet,	<aus. fast. 3.15>
IGNOTA,aeternae,aeternae	<aus. fast. 1.1><aus. fast. 1.1><aus. fast. 1.1><aus. fast. 1.1><aus. fast. 1.
IGNOTA,ne,ne	<aus. fast. 1.1><aus. fast. 1.1><aus. fast. 1.1><aus. fast. 1.1>
IGNOTA,sint,sint	<aus. fast. 1.1><aus. fast. 1.1><aus. fast. 1.1>
IGNOTA,tempora,tempora	<aus. fast. 1.1>
IGNOTA,tibi,tibi	<aus. fast. 1.1><aus. fast. 1.1>
Iulium,61.5,61.5	<sen. eld. fr. 1.4>
Iulium,Donat.,Donat.	<sen. eld. fr. 1.4><sen. eld. fr. 1.4><sen. eld. fr. 1.4><sen. eld. fr. 1.4><sen. eld.
Iulium,Montanum,Montanum	<sen. eld. fr. 1.4><sen. eld. fr. 1.4><sen. eld. fr. 1.4><sen. eld. fr. 1.4><se
Iulium,Vergilii,Vergilii	<sen. eld. fr. 1.4><sen. eld. fr. 1.4><sen. eld. fr. 1.4>
Iulium,Vergilio,Vergilio	<sen. eld. fr. 1.4><sen. eld. fr. 1.4><sen. eld. fr. 1.4><sen. eld. fr. 1.4><se
Iulium,bene,bene	<sen. eld. fr. 1.4><sen. eld. fr. 1.4><sen. eld. fr. 1.4><sen. eld. fr. 1.4><sen. eld.
Iulium,dicere,,dicere,	<sen. eld. fr. 1.4><sen. eld. fr. 1.4><sen. eld. fr. 1.4><sen. eld. fr. 1.4><sen. eld.
Iulium,enim,enim	<sen. eld. fr. 1.4><sen. eld. fr. 1.4><sen. eld. fr. 1.4><sen. eld. fr. 1.4><sen. eld.
Iulium,eosdem,eosdem	<sen. eld. fr. 1.4><sen. eld. fr. 1.4><sen. eld. fr. 1.4><sen. eld. fr. 1.4><sen. eld.
Iulium,esse,esse	<sen. eld. fr. 1.4><sen. eld. fr. 1.4><sen. eld. fr. 1.4><sen. eld. fr. 1.4><sen. eld.
Iulium,et,et	<sen. eld. fr. 1.4><sen. eld. fr. 1.4><sen. eld. fr. 1.4><sen. eld. fr. 1.4><sen. eld. fr. 1.4
[illegible]
```

Plot:

Reference: Book 1.xlsx

Steps Performed:

1. I took a number of readings and recorded the time taken by each reading. I set the number of documents as 2,4,6,8 and 10 recorded the time taken during each run. I did this by adding time before my execution command. Ex: time hadoop jar fa1.jar FeaturedActivity1 ~/LatinInputText/Downloads/2latindocs ~/fa1
2. I drew a plot of time vs number of documents processed. (For reference use Book 1.xlsx)



Observations:

1. Similar to earlier plot here also we can infer that the execution time increases as the size of input to Hadoop (number of documents) increases.
2. However, here you can see that the program being a little more complex than before, even for low values of documents, the execution time is drastically increasing. Hence we can also infer that the execution time is increasing with increase in the program complexity.