Q1. (5 pts) Consider the following fragment from a collection of documents. Assume that these three documents are the only documents in the collection that contain the words “dog” or “cat”.

Document id	**Document Text**
234569	The phrase "fight like cats and dogs" reflects a natural tendency for the relationship between the two species to be antagonistic. However, sometimes the two species can be friends.
234578	Dogs and cats can have a bad relationship.
234839	Cats are furry.
234879	Dogs are man's best friend.

1. (1 pts) Write the inverted index posting lists for terms “cat” and “dog”.

   cat : 234569:1, 234578:1, 234839:1
   dog: 234569:1, 234578:1, 234879:1

2. (2 pts) Write the compressed form of the posting list for dog.

   dog: 234569:1, 9:1, 301:1

   In binary:
   234,569 = b111001010001001001
   1 = b1
   9 = b1001
   301 = b100101101

   VB encoding:
   234,569: 00001110 00101000 11001001
   1: 10000001
   9: 10001001
   1: 10000001
   301: 00000010 10101101
   1: 10000001

3. (2 pts) Calculate the cosine similarity of each of the above documents with query
   q: cat and dogs.
   Use stemming and remove stop words from the documents and query.
   Use only TF (ignore IDF).

   We compute the document vector magnitudes as follows.

Document id	**Document Text**	Maxf	Document vector magnitudes
234569	The phrase "fight like cats and dogs" reflects a natural tendency for the relationship between the two species to be antagonistic. However, sometimes the two species can be friends.	2 (species, two)	Sqrt(phrase 1+ fight 1+ cat 1+ dog 1+ reflects 1+ natural 1+ tendency 1+ relationship 1+ two 2+ species 2+ antagonistic 1+ friend 1) = Sqrt((1/2)^2+ (1/2)^2+ (1/2)^2+ (1/2)^2+ (1/2)^2+ (1/2)^2+ (1/2)^2+ (1/2)^2+ (2/2)^2+ (2/2)^2+ (1/2)^2+ (1/2)^2) = 2.12
234578	Dogs and cats can have a bad relationship.	1	Sqrt(4) = 2
234839	Cats are furry.	1	Sqrt(2) = 1.41
234879	Dogs are man's best friend.	1	Sqrt(4) = 2

Query magnitude: Sqrt(2) = 1.41
Cos(q, d_{234569}) = (.5+.5)/(2.12*1.41) = 0.33
Cos(q, d_{234578}) = (1+1)/(2*1.41) = 0.71
Cos(q, d_{234839}) = 1/(1.41*1.41) = 0.5
Cos(q, d_{234879}) = 1/(2*1.41) = 0.35

Q2. (4 pts) Design MapReduce algorithms to take a very large file of integers and produce as output:
(a) The largest integer.
(b) The average of all the integers.
(c) The same set of integers, but with each integer appearing only once.
(d) The count of the number of distinct integers in the input.

Sol.
(a)

Map(id, i):
  emit (1,i)
	Combine(j, li):
	  emit (1, max(li))
		Reduce(j, li):
		  emit (max(li), null)

Notes. (1) The combiner will greatly reduce the number of intermediate key/value pairs sent across the network to the reducers. (2) There will be only one reducer. The length of the list in the reducer will be equal to number of map tasks executed.

(b)

Map(id,i):
  emit (1,i)
	Combine(j, li):
	  emit (1, (s, sum(li)))
	  emit (1, (c, count(li)))
		Reduce(j, li):
		  ls = [x for x in li if x[0]==s]
		  lc = [x for x in li if x[0]==c]
		  emit (sum(ls)/sum(lc), null)

Notes. Same as for point (a).

(c)

Map(id,i):
  emit (i, null)
	Reduce(i,li):
	  emit (i, null)

(d)
Use as input the output produced by (c).

Map(i,null):
  emit (1,1)
	Combine(i,li):
	  emit (1,count(li))
		Reduce(i,li):
		  emit (sum(li),null)

Q3. (3 pts) Write MapReduce algorithms for computing the following operations on bags R and S:
(a) Bag Union, defined to be the bag of tuples in which tuple t appears the sum of the numbers of times it appears in R and S.
(b) Bag Intersection, defined to be the bag of tuples in which tuple t appears the minimum of the numbers of times it appears in R and S.
(c) Bag Difference, defined to be the bag of tuples in which the number of times a tuple t appears is equal to the number of times it appears in R minus the number of times it appears in S. A tuple that appears more times in S than in R does not appear in the difference.

Sol.

(a) Bag Union

tid is a tuple id, which we don't use in this problem. t is a tuple.

```
Map(tid,t):
  emit(t,1)
```

```
Reduce(t,li):
   emit(t,len(li))
```

(b) Bag Intersection

tid is a tuple id, which we don't use in this problem. t is a tuple. X is a bit indicating relation R or S.

```
Map(tid, (t,X)):
  emit(t,X)
```

```
Reduce(t,li):
   lr = [x for x in li if x==R]
   ls = [x for x in li if x==S]
   c = min(len(lr),len(ls))
   if c>0: emit(t,c)
```

(c) Bag Difference

tid is a tuple id, which we don't use in this problem. t is a tuple. X is a bit indicating relation R or S.

```
Map(tid, (t,X)):
  emit(t,X)
```

```
Reduce(t,li):
   lr = [x for x in li if x==R]
   ls = [x for x in li if x==S]
   c = len(lr) - len(ls)
   if c>0: emit(t,c)
```

Q4. (6 pts) Implement the following tasks using RDDs in PySpark.
Consider the movielens small dataset; see: https://grouplens.org/datasets/movielens for a description of the dataset.
Here is the zip file: http://files.grouplens.org/datasets/movielens/ml-latest-small.zip

1. (2 pts) Find the average rating for each user and movie.
   Use the "ratings.csv" file.
2. (4 pts) Find the average rating for each genre.
   Use the "movies.csv" file to find the genre of each movie.
   This file is assumed to fit in the memory of a machine; it is cached locally.
   A movie can belong to multiple genres.
   A rating of a movie should be used for all the genres the movie belongs in.

Submit all your source code and your outputs.

```
#Q4.1
#filter out the first line (header line which starts with 'userId,movieId')
linesRDD = sc.textFile("ratings.csv")\
.filter(lambda line: not line.startswith('userId,movieId'))

tuplesRDD = linesRDD.map(lambda line: line.split(','))

pairsRDD = tuplesRDD\
.flatMap(lambda x: ( ('u'+x[0], float(x[2])), ('m'+x[1], float(x[2])) ) )

sumcntRDD = pairsRDD.aggregateByKey((0,0),
                                    lambda acc,rating: (acc[0]+rating, acc[1]+1),
                                    lambda acc1, acc2: (acc1[0]+acc2[0], acc1[1]+acc2[1]))

avgRDD = sumcntRDD.map(lambda x: (x[0], x[1][0]/x[1][1]))
print(avgRDD.take(5))


#Q4.2
# Let's convert the movies.csv text file into a dictionary of movieid, genre_list
import csv
datafile = open('movies.csv', 'r')
myreader = csv.reader(datafile)

movieGenres = {}
for row in myreader:
    movieGenres[row[0]] = row[2].split('|')

linesRDD = sc.textFile("ratings.csv")\
.filter(lambda line: not line.startswith('userId,movieId'))

tuplesRDD = linesRDD.map(lambda line: line.split(','))

pairsRDD = tuplesRDD\
.flatMap(lambda x: [ (genre, float(x[2])) for genre in movieGenres[x[1]] ]   )

sumcntRDD = pairsRDD.aggregateByKey((0,0),
                                    lambda acc,rating: (acc[0]+rating, acc[1]+1),
                                    lambda acc1, acc2: (acc1[0]+acc2[0], acc1[1]+acc2[1]))

avgRDD = sumcntRDD.map(lambda x: (x[0], x[1][0]/x[1][1]))
print(avgRDD.collect())
```