1. Write a spark program to count the number of words given in the file pg100.txt
   Get the file from here : https://www.gutenberg.org/cache/epub/100/pg100.txt

```
import re
import sys
from pyspark import SparkConf, SparkContext
conf = SparkConf()
sc = SparkContext(conf=conf)
lines = sc.textFile(sys.argv[1])
words = lines.flatMap(lambda l:re.split(r'[^\w]+',l))
pairs = words.map(lambda w: (w, 1))
counts = pairs.reduceByKey(lambda n1, n2: n1 + n2)
counts.saveAsTextFile(sys.argv[2])
sc.stop()
```

2. Compute Bi-Grams from the same file pg100.txt.

3. K-means implemnatation with spark:

This problem will help you understand the nitty gritty details of implementing clustering algorithms on Spark. In addition, this problem will also help you understand the impact of using various distance metrics and initialization strategies in practice. Let us say we have a set $\mathcal{X}$ of n data points in the d-dimensional space $\mathbb{R}^d$. Given the number of clusters k and the set of k centroids $\mathcal{C}$, we now proceed to define various distance metrics and the corresponding cost functions that they minimize.

Euclidean distance Given two points A and B in d dimensional space such that $A = [a_1, a_2 \cdots a_d]$ and $B = [b_1, b_2 \cdots b_d]$, the Euclidean distance between A and B is defined as:

$$||a - b|| = \sqrt{\sum_{i=1}^{d} (a_i - b_i)^2} \tag{1}$$

The corresponding cost function ϕ that is minimized when we assign points to clusters using the Euclidean distance metric is given by:

$$\phi = \sum_{x \in \mathcal{X}} \min_{c \in \mathcal{C}} ||x - c||^2 \tag{2}$$

Manhattan distance Given two random points A and B in d dimensional space such that $A = [a_1, a_2 \cdots a_d]$ and $B = [b_1, b_2 \cdots b_d]$, the Manhattan distance between A and B is defined as:

$$|a - b| = \sum_{i=1}^{d} |a_i - b_i| \tag{3}$$

The corresponding cost function ψ that is minimized when we assign points to clusters using the Manhattan distance metric is given by:

$$\psi = \sum_{x \in \mathcal{X}} \min_{c \in \mathcal{C}} |x - c| \tag{4}$$

Iterative k-Means Algorithm: We learned the basic k-Means algorithm in class which is as follows: k centroids are initialized, each point is assigned to the nearest centroid and

the centroids are recomputed based on the assignments of points to clusters. In practice, the above steps are run for several iterations. We present the resulting iterative version of k-Means in Algorithm 1.

Algorithm 1 Iterative k-Means Algorithm

```
1: procedure ITERATIVE k-MEANS
2:     Select k points as initial centroids of the k clusters.
3:     for iterations := 1 to MAX_ITER do
4:         for each point p in the dataset do
5:             Assign point p to the cluster with the closest centroid
6:         end for
7:         for each cluster c do
8:             Recompute the centroid of c as the mean of all the data points assigned to c
9:         end for
10:    end for
11: end procedure
```

Implement iterative k-means using Spark. The datasets are attached in k-means directory. In the file you will see three files:

data.txt contains the original dataset with 4601 rows and 58 columns. Each row is essentially a document that is represented using a 58 dimensional vector. Each component shows importance of a word in the document.

near.txt contains k initial centroids. They were chosen by setting k=10 random points.

far.txt contains initial cluster centroids which are as far as possible. This is done by selecting one point first at random and finding the second point which is the farthest from the first. Then the third is selected which is farthest from first, and second, and so on.

Set maximum number of iterations to 20 and the number of clusters k to 10 for all the experiments in this.

For following 2(a) and 2(b) make changes in eculidean_cluster.py file (you can use scala too if you want). Provide two clear defined functions for doing the two sub-parts. Submit a pdf with the plots and percentage improvement values as stated.

2(a) Using Euclidean distance, shown in Eq. 1, as the distance measure, compute the cost function phi() shown in Eq. 2. for every iteration i. Meaning you first need to compute the cost function for any one of the two centroid files given. Now run data.txt with near.txt and far.txt. Generate a graph for phi(i) where i=1…20 for both near.txt and far.txt.

2(b) What is the percentage change in cost after 10 iters of k-means algorithm when clusters are initialized with near.txt vs far.txt given that you are using Euclidean distance. Is the random initialization using near.txt better in comparison to far.txt? Explain.

For following 2(c) and 2(d) make changes in manhattan_cluster.py. Provide two clear defined functions for doing the two sub-parts. Submit a pdf with the plots and percentage improvement values.

2(c) Now use Manhattan distance as the cost metric as shown in Eq.3 and compute the cost psi() as shown in Eq. 4 and repeat same as 2(a)

2(d) Repeat 2(b) for Manhattan distance.