**Final Report**

***Jonathan Galsurkar & Andrew Miloslavsky***

1. Get the totalNumberOfDocuments count
2. Flatmap each document and split the words in the document by whitespace

We chose this because we realized that we can get all of the information necessary for the tf-idf calculation in one step, optimizing and preventing reprocessing of each line.

1. Filter the terms to match the gene format and map each term to the key as (term, documentID, totalWordsInDocument) , 1. This is the wordcount problem.
2. MAP : (term, documentID, totalWordsInDocument) => 1
3. We reduce this by key to get the total wordCount for that term per document
4. REDUCE: (term, documentID, totalWordsInDocument) => wordCountForTerm
5. ReMAP this such that : term => (documentID, totalWordsInDocument, wordCountForTerm) which is the rest of the info we need to calculate the tf-idf
6. With the term being the new key, we can group by the key (REDUCE) to get a list of the proper values
7. We map this such that we calculate a list of tf-idfs for each term for the proper documents
8. MAP : Term => Iterable(documentID, tf-idf)

((term1, Iterable[(documentID1, tf-idf1),…,(documentIDN, tf-idfN)]), (term2, Iterable[(documentID1, tf-idf1),…,(documentIDN, tf-idfN)]) )…)

1. We take the Cartesian of the two in order to get the pairs of terms but filter it by lexicographical difference to avoid duplicates
2. We map each pair to their cosineSimilarity, pairNames and filter out those genes with 0 similarity since they don’t give us much useful information and speeds up the program
3. MAP : cosineSimilarity => (term1, term2)
4. Sort by the cosine similarity in descending order. This also would allow us to group pairs with the same cosineSimilarity if we chose to do so.