Ahmed Alhaj
CMSC476
Phase 4

In this phase of the project, I decided to implement the vector space model. So I created another 2 structures The first vecto space model where the main search is done and the other for the calculations. The other structure is the Token vector where there will be a query vector of the term weight and document vector.

So the I re-run BSBI to build the inverted index with some modifications, so that it is pretty fit for the vector space model. I save the preprocessed data include inverted index and the postings where you have tokens with its documents and frequency. Since I am implemented the BSBI for building the index I have sorted block of retrieval documents saved. With the inverted index and normalized the tf-idf for the term weight also. So for the query search I am just loading the inverted index and the term weight and the retrieval documents block.

With the query search, the process of building the query vector goes as follows 1-tokenize the query, 2-calculate the term weight for the query tokens. the query vector uses log-weighted term frequency, idf weighting, and cosine normalization.

Each document vector is normalized by the Euclidean length of the vector, so that all document vectors turned into unit vectors. That is done by eliminating all information on the length of the original document; this masks some subtleties about longer documents. First, longer documents will have more terms and higher tf values. Second, longer documents contain more distinct terms.

$$cos(doc,\ query) = \Sigma W t,q\ W t,d / \Sigma\, w^2 t,q\ \ \Sigma t \in d\ \ W^2\ t,d$$

Computing cosine scores

```
CosineScore(q)
 1  float Scores[N] = 0
 2  float Length[N]
 3  for each query term t
 4  do calculate w_{t,q} and fetch postings list for t
 5     for each pair(d, tf_{t,d}) in postings list
 6     do Scores[d] += w_{t,d} × w_{t,q}
 7  Read the array Length
 8  for each d
 9  do Scores[d] = Scores[d]/Length[d]
10  return Top K components of Scores[]
```

So in the vector space ranking, the query is represented as a weighted tf-idf vector and then each document is represented as a weighted tf-idf vector. And now you can just compute the cosine similarity score for the query vector and each document vector. And then rank documents with respect to the query by score and return the top K = 10 to the user.

The rest of the structures is from previous phases.

With regard to the performance, on average takes constantly around 0.21 millisecond.
To retrieve a query. Obviously depends on the query length if the query ranges from 1 to 20 word the performance is very similar.