# SHORT REPORT OF MY IMPLEMENTATION:

In the indexer.py file I had parsed through the corpus file provided and created a dictionary, which has key as the word, and value was the list of the tuples like (doc_id frequency).
For the easy of reading back in the bm25.py code, I created the index.out file of the pattern:

*word1 # doc_id  frequency # doc_id  frequency # doc_id  frequency # doc_id  frequency*
*word2 # doc_id  frequency # doc_id  frequency # doc_id  frequency # doc_id  frequency*
*word3 # doc_id  frequency # doc_id  frequency # doc_id  frequency # doc_id  frequency*
*word4 # doc_id  frequency # doc_id  frequency # doc_id  frequency # doc_id  frequency*

where frequency is the number of times the 'Word' has appeared in that particular doc_id.

**In the indexer.py I take tccorpus.txt as the input and in this:**

STEP 1: Read the corpus and when I encounter # I put that inside the list of documents. Also I here update the word and tuple list dictionary with the key as the word. And value as the list of doc id and freq. When I don't encounter # then I read all the words and then put each a list of words which are present in the document id. And then I update the dictionary with doc id as the key and list of words in it as the value.

STEP 2: Now I iterate over the above dictionary and for each list of the document id I create a new dictionary for the term frequency called the "dict_for_tf" dictionary. This has word as the key and its frequency in the document id as the value.

STEP 3: Now I read back this dictionary and use the value of the term frequency from it and put it inside the tuple of doc_id and term frequency.
And now I create the list of these tuples for each doc id and put it as the value in the dictionary (inverted_index_list), which has the final inverted index data.

STEP 4: Finally I display the data (index.out file) as the format stated below:

*word1 # doc_id  frequency # doc_id  frequency # doc_id  frequency # doc_id  frequency*
*word2 # doc_id  frequency # doc_id  frequency # doc_id  frequency # doc_id  frequency*
*word3 # doc_id  frequency # doc_id  frequency # doc_id  frequency # doc_id  frequency*
*word4 # doc_id  frequency # doc_id  frequency # doc_id  frequency # doc_id  frequency*

where frequency is the number of times the 'Word' has appeared in that particular doc_id.

**In the bm25.py I take index.out, queries.txt and max_limit as the input and in this:**

STEP 1: Firstly I read the index.out file from the input and restore back the data into the dictionary (index_dict) with word as the key and value as the list of doc id and frequency tuple.

STEP 2: Now in the dl_size_dict dictionary I have maintained key as the document id and value as the length of that document. And calculated the total size by iterating over this dictionary and adding all the size to get total size of all the docs.

STEP 3: I calculated the average document length:

avdl = total_size/float(len(dl_size_dict))

STEP 4: Now I maintained a dictionary (relative_dl), which keeps track of the document id and its dl/avdl value, which is later used in bm25 calculation.

STEP 5: Created a dictionary that keeps track of the query line and its id (query_n_id_dict), with key being the id and value being the actual query.

STEP 6: Iterating over the query_n_id_dict over each of the query line from the queries.txt file and then for each query I iterate over all the documents and apply the bm25 algorithm, which is provided in professor's slide, and calculate the score for each document for the given query form the queries.txt file.

STEP 7: My code has two separate LOC for calculating the scores; one calculates the score of random 100 documents for each query and displays the max possible list (max_limit). The second LOC actually calculates the scores of the entire documents and then sorts them in reverse order and then display the top 100 (max_limit) documents and their ranks.
The output is finally displayed is the following format:

```
query_id Q0 doc_id rank BM25_score system_name
```

I choose AkshayMacBookPro as it's my own systems name.

*NOTE: For line-by-line explanations, please refer the code file comments.*