Chang Hyun Lee
N12218805
CS6913
HW03

Problem 1.

(a) Instead of scanning the whole index data structure in order to search a particular term, search engines use early termination to efficiently access the inverted list whose index matches the term or is semantically close to the meaning of the term. Thus, this technique cuts out the indices that are not relevant to the term and reduces the access time to the inverted list. The methods include but are not limited to the reduction of the index structure and the limit to the access at query time to the most promising parts of the index.

(b) Fagin's algorithm is the top-k query-processing algorithm.

There are a number of lists, and each element is the pair of an object and its score. The elements of all the lists must have the same objects. The algorithm sequentially scans the lists in parallel until there are different k objects that have been seen in all the lists. For example, when k = 3, the parallel processing keeps going on until all the three different objects (say, X, Y, and Z) are found in all that processed parts of the lists and their scores are all computed. When the algorithm finds the k different objects in all the lists, it makes the random accesses to the remaining elements in the lists whose objects are already discovered and computes the scores. Once the random accesses are finished, the scores are computed for the remaining elements. Lastly, the algorithm returns the objects with the top k scores.

(c) As the assumption is that the scores of the elements are sorted in the containers, the objects found while the data is processed in parallel are guaranteed to belong to the group of the top-k scoring objects.

(d) Both algorithms are designed to computer the top k scores without scanning all the containers and thus to reduce the time cost. But the algorithms must have the assumption that the lists to be scanned are already sorted by its scores. In reality, the containers that have the scores are not sorted by the scores, but rather by the name of the objects.
(e.g. word->{document id:socre}). So sorting by score can take significant time if we have to use the FA or TA.

(e) Tiered indexing is the technique to create the tiers of indices, with the first tier being the most "important" among the indexing terms. Unlike the traditional indexing which uses the word to identify the documents that contain the word, the tiered indexing includes other resources besides the text of the document, including the title of the page. Using this philosophy, the tiered indexing may eliminate the documents that still contain the word but may not be relevant to the contextual meaning of the query. As a result, it can help access the most relevant and important document much faster.

Problem 2.

This program enables the search of the relevant documents using the query search terms used. The final inverted index folder, which was made after the indexing process, have the following files: the list of the final inverted index files, the list of the files containing the byte offset of each word from the beginning of each inverted index file, the file containing the document hash table (document id, url, and the document length), and the lexicon file containing word id and word itself. You must put this folder along with two script files main.py and search.py under the same directory.

Once this setup is done, the following must be done.

1. Open the command shell.
2. Go to the directory where the script codes and the folder are located.
3. Run the following command "python main.py"
4. It will prompt you to answer some of the questions
    - what do you want to look for? (the query search terms)
    - how many documents do you want to see? (top-k documents)
    - type the number of documents to be less than 20 (IMPORTANT!!!!)

The way the program run is as follows. If the user types the query, the query is normalized (lowercase, no special characters, etc), and split into terms. Since the final inverted index files are named as 'start word'-'end word', all that has to be done is to check whether the particular term is within the range of start word and end word. If not, it moves to the next filename and compares the range. After finding the right file to read, it also read the right byte file. The byte file is used to find the offset of the word term from the beginning of the final inverted index file. This way eliminates the need for the file to be scanned in order to find the right term. After finding the term in the inverted index file using the byte information, the program finds the line that contains the inverted list associated with that particular term and forms the inverted index that contains only the terms of the query.

After the inverted index is formed, the DATT technique is used to retrieve the top k scoring documents. And the scoring function is BM25. Unlike the instruction, I did not create openList() or closeList(), as the inverted index that contains the needed terms is already formed and Python uses garbage collection that removes the in-memory data once the program is completed. On average, one search term for retrieving 10 relevant documents takes about 0.19 second. Below is the result when typing 'love' and retrieving top 10 documents.

```
172-16-20-5:hw03 chlee021690$ python main.py
===============SEARCH===============
52065
what do you want to look for? love
how many documents do you want to see? 10
['http://images.recoil.net.nz/view/3830', 1] 12.2138696429
['http://www.pottersmarks.co.nz/potlane/pages/Crewenna%20Pottery.htm', 12] 11.9050920395
['http://www.poolsafe.org.nz/public_pools/supervision.html', 34] 11.416652706
['http://www.polytechnic.ac.nz/programmesandcourses/courses/cm/74305.html', 35]
11.3954014393
['http://www.placestostay.co.nz/places/5343.asp', 39] 11.3111816791
['http://www.placestostay.co.nz/places/5273.asp', 39] 11.3111816791
['http://www.polytechnic.ac.nz/programmesandcourses/programmetypes/certificates/op5266.ht
ml', 42] 11.2488293097
['http://www.pixelpark.co.nz/war_games/vinci.htm', 44] 11.2076415702
['http://www.pricespy.co.nz/gid_170.html', 54] 11.0061458168
```

```
['http://www.peterellis.org.nz/ACC/2002/2002-0107_Wakefield_Flyer.htm', 62] 10.8500917603
run-time : 0.19414 seconds.
```

The files tested were the final inverted index files from NZ2 dataset. The size of the index file was about 30970578 bytes. To pick out the particular module/functionality, I would indicate the use of heapq, the Python library for heap data structure. This was necessary to be used, because binary heap is the efficient data structure to access and store the top k scoring documents.