Programming Assignment 1

(Team 3)

Team Members:

	Names	UNH ID	Webcat Username	Assignment No
1.	Amith Ramanagar Chandrashekar	928101607	ar1184	1
2.	Vaughan Coder	966427992	vwc1010	1
3.	Sepideh Koohfar	928334858	sk1015	1
4.	Pooja Himanshu Oza	900539783	pho1003	1

Code Explanation:

1. The program takes the paragraph file path and the index directory path as input.
2. Once we get the input, we start to build the Lucene index. For building the index, we first configure the IndexWriter. IndexWriter is responsible for creating and maintaining the Lucene index. The directory path where the index needs to be stored is provided as an input.
3. While creating IndexWriter object, we need to specify Analyzer that Lucene should use while creating the index. Analyzer provides a means to convert the given string or reader into tokens which is used by Lucene for indexing and searching. Here we are using StandardAnalyzer, which breaks the text into tokens using Word Break rules, lowercases the tokens and filter out the stopwords.
4. After the IndexWriter object is instantiated, we now read the input paragraphs file using iterableParagraphs method of TREC-CAR-tools. IterableParagraphs method reads the given cbor paragraphs input file and provides the paragraphID, paragraph text and paragraph links as output. Here we are concerned only with the paragraph ID and paragraph body.
5. As we loop through the paragraphs, we add the content and ID of each paragraph as 1 document in the index. We are storing paragraphID as StringField and paragraph content as TextField. We are storing the paragraphID as StringField since we need to store the value as a single token and not tokenize it.
6. Once we add all the documents and close the Lucene Index, we start with the searching. For searching, we create a query object for each input query. To create the query object, we use QueryParser which parses the input query. To parse the query we are using the default behaviour of split on whitespace and the default conjunctive operator OR. Each token is taken as 1 term in the query.
7. Default Search: Once the query is parsed, we create an instance of IndexSearcher and search the index for the given query using the default ranking function i.e. BM25. Okapi BM25 is a ranking function based on probabilistic relevance framework. Okapi BM25 formula consists

of term frequency (TF), inverse document frequency (IDF), average document length and several other parameters.

8. Custom Search: In custom search, we create an instance of IndexSearcher and override the method score of the Similarity class. In this search, we are considering the total term frequency of the query terms in each document and ranking the document accordingly.
9. Lucene uses the above search formulas to rank the documents and return the results.

Scoring function:

According to the Lucene documentation, BM25 is being used as the default Ranking function.

The formula used for Lucene BM25 is

$$BM25 = \sum IDF(qi) * (f(qi,D) * (k1+1)/ f(qi,D)+ k1*(1-b+b*(doclen/averaged)))$$

$$\text{Where } \sum IDF(qi) = \log(1 + (N - n + 0.5) / (n + 0.5))$$

Results:

Lucene default search (BM25 Similarity)

1) power nap benefits

ParagraphID	Score
85bcaa2516682b1738c121bfd1d7bd60c9d2e274	19.193037
76cae6cb9749c647ae52077d6fd535f3ccdb41a2	16.194584
05ee98915108d6fea8b95d4aefd51acadf85bb3a	15.985866
9fe0ea9205e708269ec2cf437aa23360c5805a8b	15.64356
0a0af8bdfc8a4ead32792ccd702dd6455e068d16	11.960827
1b470a36adea668e666acefd8b82ba1336620315	11.588797
0bb27470730936e60db6de54836ef6700c58e53f	10.416327
bf2d3d9fab4e234bcf4ce753f9e99a8c90e9cea9	10.33079
31b12608564134c2d86ad73ed53f5ad7997f1caa	9.972311
857c9393cc9f1438f3dc5a08f512226abc414e87	8.707433

2) whale vocalization production of sound

ParagraphID	Score
8e2821b4d1948204788a311bb15a0989577aa8df	17.34641
35a73dca142e7e4ceb6716d9583b3486e2c19051	15.312756
52d1827627d2fdb8271eed24f71a424769595951	14.974876
750b53d8441e81fd9f87e3a41dee7c8fdd7be9ca	12.816278
fbff039e5c107c9f8be00da48add3995428773d7	12.066385

a3c550c198a78e23bbee44b25db74e7b743b573e	9.984061
711eb45bef2339ddc2cd090e60899c756feb493d	9.5355
c4c746a886f0c06688bad3d9419cd5fb56fc1ffb	8.75445
776ad7a0fdb5a50aac95f0a44468aa6a5310a41b	8.699595
33f7ccf51ab0c242135ce906c6a26328a17d9308	8.659395

3) pokemon puzzle league

ParagraphID	**Score**
80f928fd3ba87a70411de560d51b93abf2c6bb66	17.444939
5df575da5cd13dd1d045119ae9aef434c7875707	16.808868
4a98bf4038f1cb4bf44e91953a52bd51f6c527aa	12.149624
29495dcc618b43427fd2f5920a5dc9decce54049	10.896196
3f28912fb9c6b2fa4377414a348275e59b7d90f5	8.570509
c1ebc5e2ad12505f150b8949a56c774bb7720183	8.2282915
122d144c144f20998d6c8a48c91a0af7bc81ac04	7.383132
6e296b55cad6d942cef7e68097d2c29eb6d446c6	7.2969847
d75ff7b28136f10a94889bcdf2f6c3200dcc3e7f	7.2213683
6d3fff9a7a74078c5b68a6f63e1fc7691f81d50e	7.2128243

Custom search:

1) power nap benefits

ParagraphID	**Score**
9fe0ea9205e708269ec2cf437aa23360c5805a8b	11.0
31b12608564134c2d86ad73ed53f5ad7997f1caa	9.0
76cae6cb9749c647ae52077d6fd535f3ccdb41a2	6.0
ce4a8c314abedae1def1b2ae93655f43dc902717	6.0
0bb27470730936e60db6de54836ef6700c58e53f	5.0
1b470a36adea668e666acefd8b82ba1336620315	4.0
a39ab68fcfbaa9baad1533a5b2ea6d0f9bff5744	4.0
bf2d3d9fab4e234bcf4ce753f9e99a8c90e9cea9	4.0
0567e33b74e0d52f31987415dc93ed41816bb4cd	3.0
05ee98915108d6fea8b95d4aefd51acadf85bb3a	3.0

2) whale vocalization production of sound

ParagraphID	**Score**
52d1827627d2fdb8271eed24f71a424769595951	6.0
fe53859fc48049c4024adff3d175fca54f84b6e4	6.0
35a73dca142e7e4ceb6716d9583b3486e2c19051	5.0
72644204bbed7fe9b443bcf12052684f12c93374	5.0
750b53d8441e81fd9f87e3a41dee7c8fdd7be9ca	5.0
a83dbce81b525478ec6111e959f4e24f0040289f	5.0
fbff039e5c107c9f8be00da48add3995428773d7	5.0

064722811a80b660bd940184e7ac3d1629334f39	4.0
44dbf787f368875787080558435c0392b99b8357	4.0
8a370428f3b085e0d11c8c6390aef42536fc7337	4.0

3) pokemon puzzle league

ParagraphID	**Score**
80f928fd3ba87a70411de560d51b93abf2c6bb66	6.0
5df575da5cd13dd1d045119ae9aef434c7875707	4.0
d75ff7b28136f10a94889bcdf2f6c3200dcc3e7f	4.0
b8d505b181ac086b69cea67a65ba517491bd34e0	3.0
1513e4cccf389f00cf99bcbe47bcb5dbfe9d5999	2.0
29495dcc618b43427fd2f5920a5dc9decce54049	2.0
3f28912fb9c6b2fa4377414a348275e59b7d90f5	2.0
40beb662b5ec81b519747c14fde3d23e746b7ba5	2.0
4a98bf4038f1cb4bf44e91953a52bd51f6c527aa	2.0
97924bab16d053e96ee70690b893b32559be8fa3	2.0

Observations:

- On observing the results of both default Lucene scoring function and custom scoring function, the team found that there are several irrelevant paragraphs which were scored higher than the relevant ones in both the scoring functions.
- In particular, the custom scoring function had several paragraphs with higher scores which were not at all relevant to the given topic. For example, the results of the query ‘power nap benefits’ had paragraph about ‘solar power plant’ with high score as the word ‘power’ had higher frequency in the paragraph.
- On comparing the results of default Lucene scoring function and the custom scoring function, it was observed that the results produced by the default scoring function were in general more relevant than the custom function.