Report Project 1: YouGle (You First Search Engine)

Member 1: Dujnapa Tanundet	ID: 6088105	Section: 1
Member 2: Klinton Chhun	ID: 6088111	Section: 1
Member 3: Arada Puengmongkolchaikit	ID: 6088133	Section: 1

Q1: A brief description of how your program is structured, and how the key steps in your indexing and retrieval algorithms work. Make sure you report statistics on the size of the index and statistics of retrieval time for the development queries.

- There are 7 java files in our default package which are BaseIndex.java, BasicIndex.java, Index.java, P1Tester.java, Pair.java, PostingList.java, and Query.java. All these files are instructed as follow:
  - BaseIndex.java: is the interface of BasicIndex.java.
  - BasicIndex.java: is the class that implemented from BaseIndex.java. This class is used for writing and reading posting list which is consisted of term id, frequency, and the list of the document id.
  - Index.java: is the class which is used for indexing method.
  - P1Tester.java: is just a tester class.
  - Pair.java: is just a class that maintaining the pair of the object.
  - PostingList.java: is the class which is used for keeping the term id and the list of document id that the term occurred.
  - Query.java: is the class that contains retrieving function retrieves the token query from the users.
- For key step of indexing, we started with the index.java file which is used the Block Sort-Based Indexing (BSBI) algorithm. After tokenized the word from files, we created postingList TreeMap, and then check the term in dictionary whether the term exists or not and then put it in postingList as term id and document id. Since we used Treemap to construct this postingList so that the list is already sorted. Then, we merged index for all postingList together.
- In the retrieving part: after we received the token or query from the users, we tokenized it. Then, we read posting list from the files. We found each query for each word by pulling out the posting list and intersected all of posting lists. Then we returned that intersected lists. If there is no matched document, we returned string "no results found".
- The statistics of index's size and retrieval time for the development queries.

Index

Index's size	small	large	citeseer
Total Files Indexed	6	98998	18824
Memory Used (MBs)	2.641776	237.210064	547.397928
Time Used (secs)	0.075	335.319	196.438
Index Size (MBs)	2.17437744140625E-4	55.37303924560547	64.60990905761719

Query

Index’s size	small	large	citeseer
Memory Used (MBs)	1.048568	168.02348	69.206016
Time Used (secs)	0.015	8.213	0.444

Q2: Answers to each of the following questions

a) **We asked you to use each sub-directory as a block and build index for one block at a time. Can you discuss the trade-off of different sizes of blocks? zIs there a general strategy when we are working with limited memory but want to minimize indexing time?**

- We are encouraged to use block sort-based indexing as well as the other data structure that map the term and the number of that term. Thus, the benefit that we got from indexing with the small size of block is that constructing the inverted index for each block or sort them much more faster, eaiser and less time. And the benefit we got from indexing with the large size of block is reducing time for merging.
- Yes, there is a general strategy when we are working with limited memory and want to minimize indexing time.

b) **Is there a part of your indexing program that limits its scalability to larger datasets? Describe all the other parts of the indexing process that you can optimize for indexing time/scalability and retrieval time**

Absolutely Yes, because we used block sort-based indexing so that we are able to create only one block for one subdirectory. If that subdirectory has too large size, this program cannot construct all the term in one block. Therefore, we are supposed to change the algorithm as constructing the block as large as the memory can store, this will reduce the merging time as well as optimizing for indexing time. For retrieval time, it is supposed to have some data structures that are able to store group terms together so it may reduce the retrieval time.

c) **Any more ideas of how to improve indexing or retrieval performance?**

There are a few ideas that we think that it can be used to improve indexing or retrieval performance. The first one is we got it from extra credit optional02 Ranked Results. This idea means that we need to define the ranking value to the query result and give the higher rank to the documents that are more relevant so that user will get the documents which are related to them efficiently.

The second idea is using another indexing algorithm. The Blocked Sort-Based indexing (BSBI) that we used in this project is excellent scaling properties. However, for the very large collections, this algorithm does not fit into memory because it needs a data structure which is mapped the terms to termIDs. Thus, the more scalable alternative than BSBI is Single-pass in memory indexing (SPIMI). Instead of using termIDs SPIMI use terms to write each block’s dictionary to disk and start new dictionary for the next blocks. Anyway, SPIMI can index any size of collections as long as the disk has enough available space.

End