Query Engine Design Spec
————————————

(1) *INPUT*

Command input:

./query [INDEXER_DATA] [CRAWLER_DIRECTORY]

Example command input:

query ../indexer/index.dat ../crawler/src/testdir

[INDEXER_DATA] ../indexer/index.dat
Requirement: The file must exist and each line should be formatted in the following way: [word] [number of documents the word occurs in] [document number] [frequency in document] [document number] [frequency in document]….
Usage: Query engine needs to be able to tell the user if the file does not exist and/or if it is formatted improperly

[CRAWLER_DIRECTORY] ../crawler/src/testdir
Requirement: The directory must exist and contain documents that correspond to the documents represented in the INDEXER_DATA file. The files should be named with unique integers, have a url on the first line.
Usage: Query engine needs to be able tot ell the user if the directory does not exist.

User input:
Once prompted, the user can input as many words as they want in the command line. The operators AND and OR can be used to interpret the query in a specific way. If there is no operator between two words it will be interpreted as an AND. The AND operator takes precedence to the OR operator. Only the first 1000 characters of input will be processed by the query engine.

(2) *OUTPUT*

A list of documents that satisfy the user's query will be printed to standard in in the format Document ID: [docID] URL: [url]. If there is nothing that satisfies the query, the user will be informed. The urls will be printed in order of relevance. Relevance is calculated by combining the frequencies of all of the words in the query that occurred in the same document.

(3) *DATA FLOW*

The file with the indexer data is read in, and the data is stored in a HashTable data structure.

The user will be prompted to enter a query which will be processed when the user

presses enter.

The first 1000 characters of the input will be stored in a string which will be parsed with spaces as delimiters.

For each word in the string, the list of DocumentNodes associated with it in the HashTable will be retrieved.

A new singly linked list of DocumentNodes will be created according to the words and operators included in the query by comparing each element of the corresponding DocumentNode lists. If two lists are being “ANDed”, the intersection of the two lists will be taken, and the rank of each document added will be calculated. If two lists are being “ORed”, the union of the two lists will be taken, and the rank of the matching documents will be calculated (non-matching documents’ rank is simply their frequency).

The final list of DocumentNodes will then be sorted by rank, and traversed to print out the document ID, and url. The url is found by opening the file in the crawler directory with the corresponding file name. If the file does not exist or cannot be opened, the user will be alerted, but the program will continue to run.

(4) *DATA STRUCTURES*

HashTable structure has an array of void pointers

HashTableNodes store WordNodes which point to lists of DocumentNodes used

DocumentNode structure has a frequency, doc_id, and next pointer (singly linked lists of these structures are used a lot)

(5) *Indexer PSEUDO CODE*

// read the indexer data file into a HashTable struct

// while the EOF character has not been inputted by the user (cntrl-d has not been pressed)

    // save the user input as a 1000 character string

    // create a list of DocumentNodes that represent the query

    // sort the list

    // report the information stored in the list