# CSCI 6370 IR and Web Search Project

## Part 2: Inverted Indexing and Query Processing

### Due date: Monday, June 15 @ 11:59 PM.

### Note 1: The course project “MySearchEngine.com” is composed of 5 part with one part per week.

### Note 2: The project is a group project. That is, you can organize a group of at most 3 members to work together.

### Note 3: Only Python is allowed for completing the course project.

## Problem Description:

This part of the project is composed the following tasks:

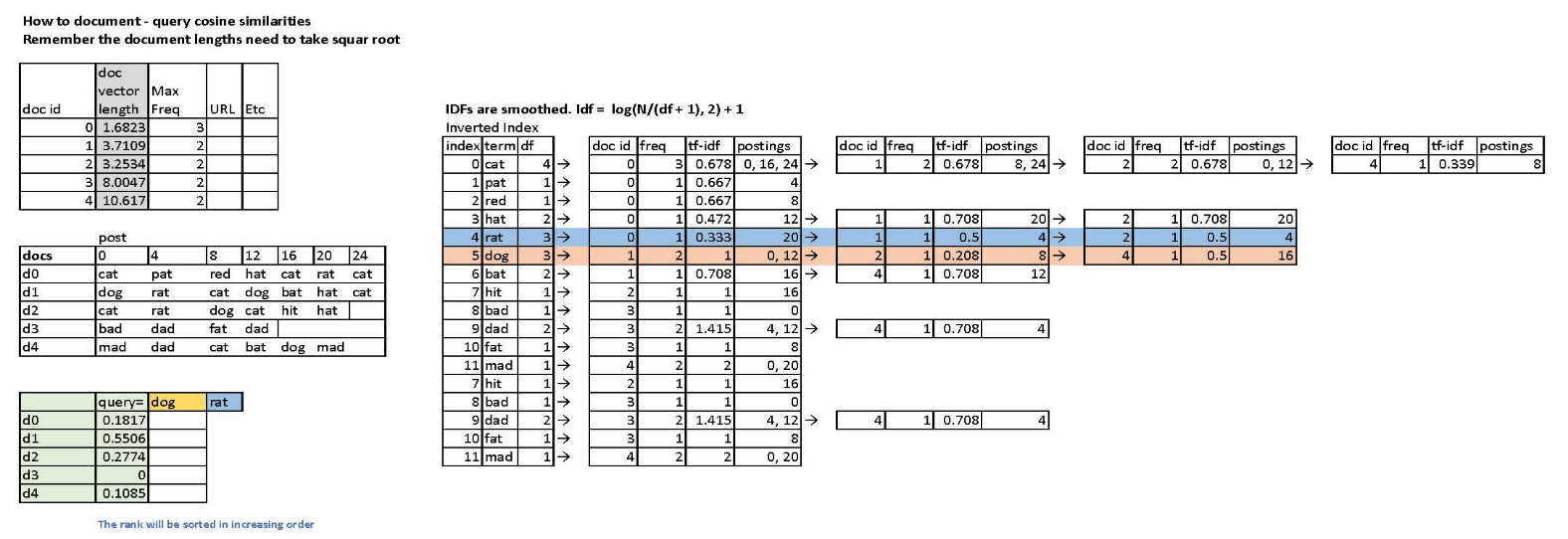
### Task 1: Build an indexer

With the completion of Part 1 of the Project, I ask you to complete the following:

* A tokenizer: to tokenize web pages. This shall include ignoring html tags, non-textual contents such as images. For textual content, it shall extract all words. It shall also extract all urls of hyperlinks contained in the web page.
* A stop-words remover: You define a list of stop-words such as “a”, “an”, “the”, “of”, etc. This part shall help remove all stop-words from those obtained from tokenizer.
* (Hash map) A document list. This shall record a document ID, document length, and maybe additional information. You can use the url of the web page as its ID, or us a unique integer for the ID but also record the url to refer to the web page.
* (Hash map) An inverted index. This index is based on the vector space model. It is a hash map of the following element type:
  + Word, document frequency of the word, and a document posting list that records all the web pages containing the word
  + A document posting list is organized as a list of records.
  + A record is composed of
    - A document ID for the web page containing the word
    - A term frequency for the frequency of the word in the web page
    - Normalized tf-idf value for the word
    - A set of positions of the word appearing in the web page
    - Maybe additional information of your choice
* A list of urls of hyperlinks extract by the tokenizer. This list shall be expandable to record the status of the urls when the web crawler is implemented in Project C: whether they have been visited or are already in the frontier url queue.
* Additional data structures of your choices.

Use the following the collection of web pages to help build and test your indexer: “Jan.zip”.

Below is an example which may help you design your inverted indices:



### Task 2: Build a simple graphic interface

The interface shall be able to

* allow a user to enter a query
* extract the query and search for documents relevant to the query, and
* return the search results to the user.

### Task 3: Build a query searcher

This task is to build a searcher for processing queries of one or more words. Below is a list of typical queries:

1. Q1= cat or dog or rat
2. Q2 = cat and dog and rat
3. Q3 = cat but dog
4. Q4 = cat dog rat

For ease of the implementation, we do not consider more complex structures of queries. The first three queries are Boolean type queries, and the fourth one is not. Their meanings are as follows:

* Q1 indicates to find documents relevant to “cat”, or “dog”, or “rat”.
* Q2 indicates to find documents relevant to “cat” and “dog” and “rat”.
* Q3 indicates to find documents relevant to “cat” but not to “dog”
* Q4 indicates to user vector space model to retrieve relevant documents.

With the help of the inverted indices completed from Task 1, the query searcher will search for relevant documents, as explain in the following:

* For Q1, search for a set A (or B, or C) of documents relevant to “cat” (or “dog”, or “rat”). The union of A, B, and C will be the answer.
* For Q2, search for a set A (or B, or C) of documents relevant to “cat” (or “dog”, or “rat”). The intersection of A, B, and C will be the answer.
* For Q3, search for a set A (or B) of documents relevant to “cat” (or “dog”). The difference A – B will be the answer.
* For Q4, just use vector space retrieval.

**Note**: For Q4, we use cosine similarity to rank documents. For Q1, Q2, and Q3, we can also us tf-idf values to rank documents as well.

### Task 4: Phrasal Search

Consider that the user enters a query:

* Q = “information retrieval evaluation”

Here, the double quotes mean that the user wants to search for documents containing the phrase “information retrieval evaluation”.

For ease of implementation, we may ignore cases of letters in the query phrase. You need to expand the ability of your query searcher to enable it to work with phrase search. You need to use the postings of words in the inverted indices to implement the phrasal search.

## How to submit your work?

Submit your python codes to Blackboard before the deadline.

One group only needs to submit one copy. Please make sure to include group members’ names and ID’s in your submission.

## Grading

The TA would do test runs of your program to check whether it works.