Fast **IN**dexing into **D**atasets (FIND)
[6.905 2015S]
[Abdi Dirie, Jason Tong]

Project overview and objective

The motivation behind FIND stems from a desire to parse and search the Child Language Data Exchange System (CHILDES), a corpus commonly used to research child language acquisition. The dataset of conversations consists of structured data that lends itself to systematically parsing tagged fields annotating the context, syntactic structure, accompanying action, and other information attached to a particular utterance in a transcript.

An extensive command line system exists for browsing the dataset online, but we wanted to create a generalized system for searching text documents that could be easily extended to the same degree of specialization as that of the existing tool, but not only for CHILDES but other similar corpora as well. There are many other bodies of well-structured textual data that does not necessarily follow precisely the same format from article to article (Wikipedia immediately comes to mind as a well-known example) and in today's data-centric world, we believe that the number of corpora that fit this bill will only grow.

One additional feature we found to be lacking in the existing tool for searching in CHILDES is a ranking metric. With such a large corpus of data, the inability to rank or in some way order positive results is a significant drawback that FIND seeks to address.

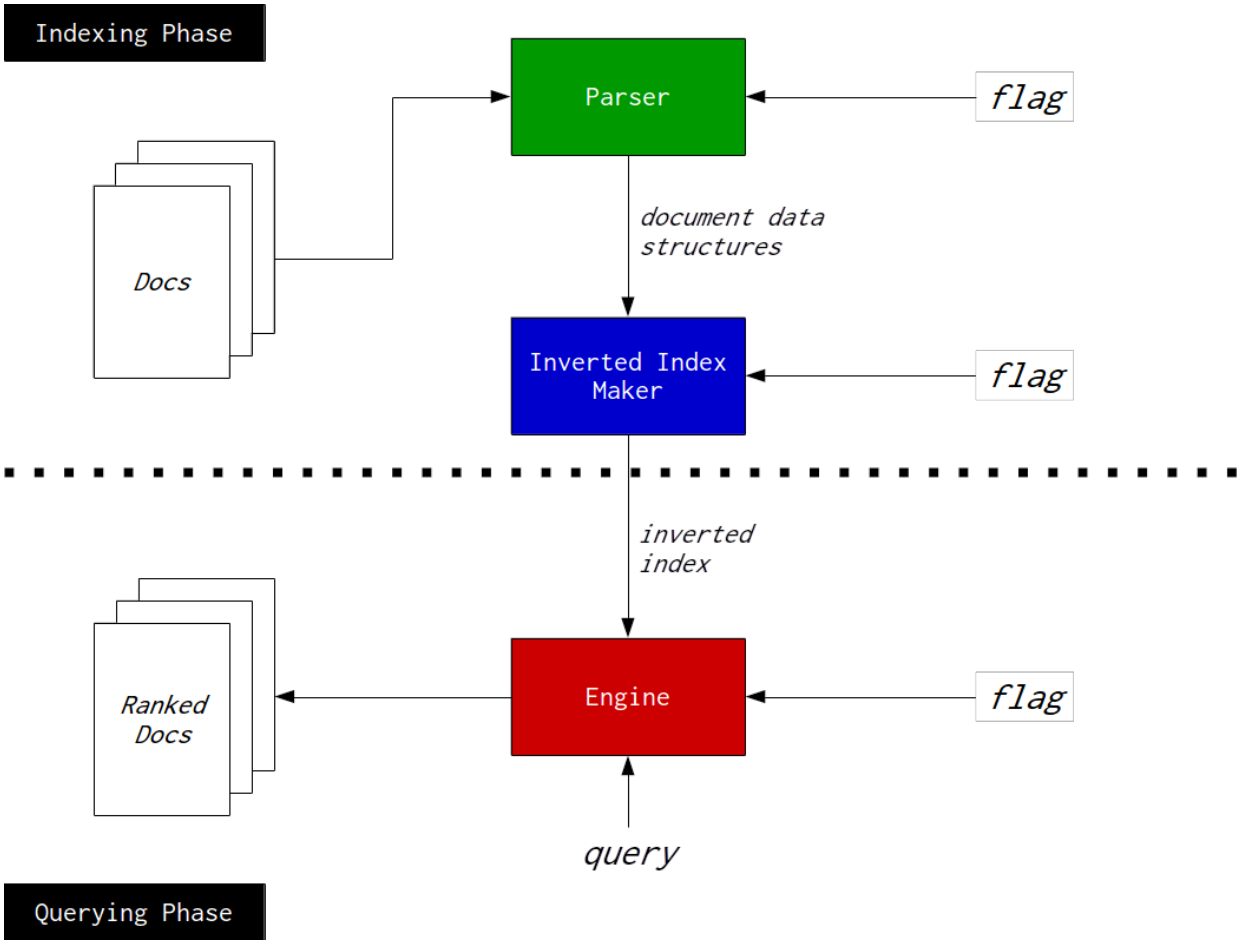
System Overview

The system has two main phases: the indexing phase and querying phase. The indexing phase starts up the system by creating all the necessary data structures. During the querying phase, the user can interact with the system to get results.

First, the indexing phase. The system ingests a corpora of textual documents and feeds it into the parser. The parser forwards its results to the inverted index maker, which then outputs the primary data structure used by the engine.

In the querying phase, the user provides a well-formed query to the system. The engine uses the passed in inverted index to best find the documents that the user query asks for. The engine returns a subset of the documents, ranked from most relevant to least relevant.

The flag parameter specifies which corpus we are dealing with, and, consequently, how the three main modules - parser, inverted index maker, and engine - execute their computations.



Modules

The system has three main modules: the parser, the inverted index maker, and the engine. To illustrate the behaviour of each component, we will use plain text documents as the corpora.

Parser

The job of the parser is to transform a textual document into a temporary in-memory data structure that is used to retrieve pieces of the document efficiently. The parser does this computation with not one document, but a batch of documents. The parser receives a list of document names and their paths. For each document, the parser return a list data structure that represents the document and its contents. For example, the parser would output the following for the document “republic.txt”:

```
(plain “republic.txt” (... “the” “Piraeus” “with” “Glaucon” “the” “son” “of” “Ariston” ...))
```

The returned object is a tagged data structure where the first element is the type of document (as a symbol), the second element is the name of the document (as a string), and the third element is a list of the words (each a string) in the order they appear in the document.

The parser can be augmented to parse a different document if it knows its structure as indicated by the flag.

Inverted Index Maker

The inverted index maker is tasked with creating an inverted index-like data structure. The inverted index is used by the engine to provide fast lookup for certain pieces of information within a specified corpora. In its simplest form, the inverted index is a hash map that maps individual words to a list where each element is a pair of the document's name and a list of the word's index in the document. For example, querying for the word "fantastic" will return:

```
(("republic.txt" 4954 104681)
 ("sherlock.txt" 17138 23346 38866 40055 42641)
 ("time.txt" 31504))
```

Extensible versions of the inverted index may use more than just words as keys and the values may be more than just association lists. As documents become more structured, nested inverted indices can be constructed within the inverted index maker so that structured information can still be looked up quickly.

Engine

The engine takes on the task of compiling the desired results of a user query made faster with the inverted index maker. It is the component that looks up the relevant documents in the corpus, and then applies the appropriate operations to retrieve the documents of interest in a ranked order.

The simplest example of a search query would be a single keyword search, where the ranking of the document might be the the frequency with which the searched word appears, or, alternatively, the term frequency-inverse document frequency (tf-idf), which is a slightly more nuanced metric that considers the "significance" of a word and takes into consideration the prevalence of a particular word in the entire corpus. The search infrastructure is designed in such a way that different ranking models could be set and swapped with ease, possibly depending on the type of data in the corpus. One way of tagging the result with its score might be to follow the name of the document with the score. In this example, word frequency is the metric for relevance:

a search for the keyword "matter" ranked by word frequency

```
(s:rank (s:keyword "matter"))
```

```
yields the following result:
(("time.txt" 4 (116 3174 13879 25815)) ("alice.txt" 2 (5856 18623)))
```

No basic search engine would be complete without the capacity to combine searches. For example, we might be interested in matching multiple keywords and be only interested in the presence of each word in the document. We might make a stricter demand that the keywords appear consecutively as a phrase. We might string together a series of predicates with boolean operators to find documents matching a very specific set of criteria. These queries of unbounded complexity are achieved with set operations on the association lists returned by the inverted index for individual keyword matches in the corpus. Of course, further processing will often have to occur once the inverted index is used to narrow down the number of potentially relevant documents in order to handle more intricate relationships between the constituent keyword searches.

Future work

CHILDES

As mentioned in the introduction, the original inspiration for FIND is derived from the recognition that there is a lot of power in a general search engine that could be extended to take advantage of structure in the data.

A particular use case for CHILDES might be a search for children who overregularize for verb tense morphological inflection. Children initially learn irregular forms well because there is no generalization mechanism and the production is very much tied to input from the environment. However, children learn morphological rules, such as adding “-ed” to produce a past tense form. At some point, children *overregularize* and produce erroneous constructions that they were unlikely to have picked up from the environment, such as “goed” or “runned.” A researcher interested in this phenomenon would be greatly facilitated by FIND if it could provide a way to search for utterances produced by children within a particular age range (header data that we can access thanks to special handling in the parser and inverted index maker) and within those, words that have been tagged as verbs tagged with past tense inflection, and even with the additional condition that it end with [d] in the phonological analysis of the utterance.

The base level search functions can be augmented to include searches that are specifically dedicated to particular annotation fields attached to utterances, also known as *tiers*. For example, we might tackle the above use case an intersection of documents with children under the age of three and documents where there is a child-produced utterance tagged as a verb in the past tense in the %mor or morphological tier (e.g. |go&PAST V|). The possibilities are unbounded.

Wikipedia

Wikipedia is a treasure trove of structured data attached to documents. Though not in the same manner as CHILDES transcripts, which contain blocks of data (utterances) each annotated with the same fields, Wikipedia has a lot of structured data attached to the entity that is the subject of a given page, such as birthplace, chemical symbol, geographic coordinates, or alma mater. Recognizing this and extending the parser and inverted index maker to handle additional specificity over where a word appears significantly reduces the search space for matching a particular query.

CHILDES and Wikipedia are only two examples sources of partially structured textual data that can benefit from the extensibility of the base level search functions offered by FIND. The entire purpose of FIND was to provide a general search engine that could then be extended to take advantage of structures in the corpus that provides more information than what is readily available from treating the entire document as a list of strings. As such, there many directions in which FIND could be extended, depending on the nature of the corpus of interest.