HiLCoE School of Computer Science

Documentation on Guragigna Information Retrieval System Project

Group Members ID

1) Meskerem Bassazin RM355

2) Nahom Gebeyehu XU0160

3) Nohe Abdella UW0395

4) Usra Mohammedseid RM355

# Introduction

Guragigna Information Retrieval System Project is a search engine written in JavaScript that has a Guragigna poem corpus and does lexical analysis. It has subsystems for stoplist, stemming, weighting, storage (indexes), indexing, query acceptance, matching, ranking and display. The project aims to provide an efficient and effective way of searching for Guragigna poems by using techniques such as lexical analysis and stop list. It allows users to search for poems based on keywords or phrases.

This project was done by consulting native speakers of the language. Guragigna is a Semitic language spoken by the Gurage people in Ethiopia. It belongs to the Southern subdivision of the Ethiopian Semitic languages within the Afroasiatic family. The Gurage languages are a subgroup of the Ethiopian Semitic languages within the Semitic family of the Afroasiatic language family. Guragigna and its dialects belong to South Ethiopic. All the Gurage subgroups (Northern, Western, and Eastern Gurage) belong to the South Ethiopic. East Gurage is related to Harari and Gafat, while Northern and Western Gurage are related to each other. The languages are often referred to as Guraginya or Guragegna. There are three dialectically varied Gurage subgroups: Northern, Eastern and Western. There are multiple variants of this language, but the one that was used for this project is Sebat bet gurage, specifically from the “Enemor” tribe. The Sebat bet gurage variant of Guragigna is spoken in the central part of Ethiopia and is one of the most widely spoken dialects of Guragigna. It is also known as “Guragegna” and has been used as a literary language since the 19th century.

Guragigna shares the same alphabet with Amharic, which is also a Semitic language. However, Guragigna is different from Amharic in almost every aspect. It has different affixes and grammatical rules. Unlike some other languages in Ethiopia, like Afan Oromo, Guragigna requires a transliterator. A transliterator is a tool that converts text from one script to another. In the case of Guragigna, it is used to convert the Guragigna script to the English script.

# Description of the text preprocessing tasks

Text preprocessing is an important task in information retrieval (IR). It involves several steps that are designed to clean and transform raw text data into a format that is more suitable for analysis. The goal of text preprocessing is to extract meaningful information from text data and to make it easier to analyze.

The text preprocessing tasks include tokenization, removing frequent stop words, stemming, and indexing. Tokenization involves breaking down text into words or other meaningful units of text. Removing frequent words involves removing words that occur frequently in the text but do not carry much meaning. Stemming involves reducing words to their root form. Indexing involves selecting terms to represent a text and finding a measure of similarity between two text representations.

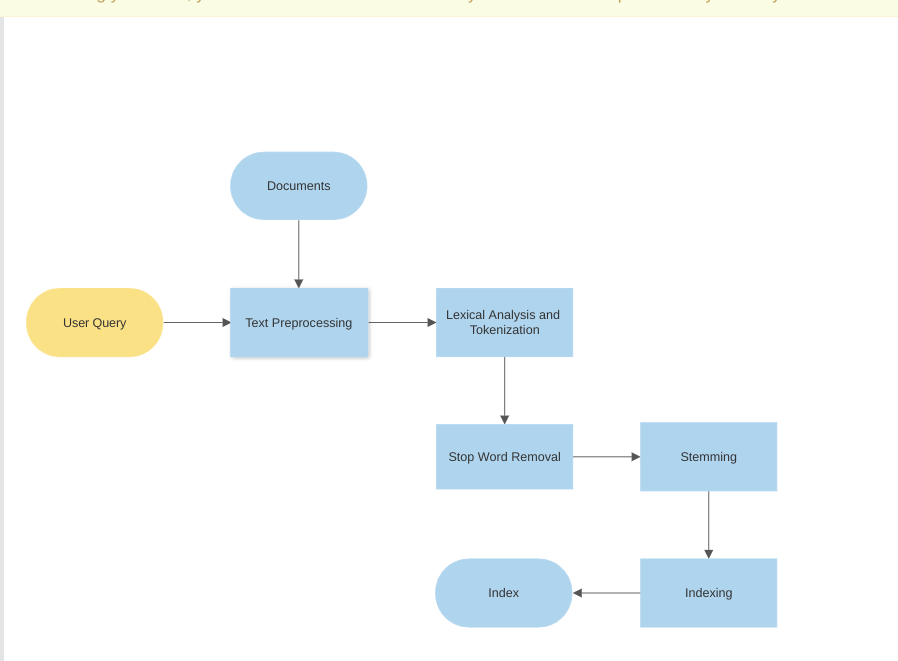
In the context of your project, tokenization involves breaking down text into words or phrases called tokens. Tokenization is an important step in information retrieval because it helps to standardize the text data and make it easier to analyze. In our project, tokenization occurs in the form of removing punctuations like “።” and “፣”, and removing stop words that occur frequently in the documents. This is done by preparing a stop list and those words will not be indexed. Removing these words can help to reduce the size of the vocabulary and improve the accuracy of the analysis.

In the context of our project, stemming involves reducing a word to its root form by removing suffixes and prefixes, generally termed as affixes. Your project doesn’t use a stemming dictionary but iteratively removes suffixes or prefixes. The stemmer doesn’t stem words that are less than 4 characters and uses an exception list to avoid stemming certain words which would cause issues and loss in semantics if stemmed. Stemming can help to reduce the number of unique words in the corpus and improve the accuracy of the analysis. However, it is not always perfect and can sometimes result in words that are not actual words or words that have lost their original meaning.

Indexing is the process of assigning a unique index to each word in the corpus. This index is used to retrieve the documents that contain the query terms. Indexing can help to speed up the search process and make it more efficient. In our project, indexing involves creating an inverted index that maps each word in the corpus to the documents that contain it. The inverted index is a data structure that stores a list of documents for each word in the corpus. This allows for fast retrieval of documents that contain specific words. The indexing process involves several steps such as tokenization, stemming, and stop word removal. After these steps are completed, the remaining words are indexed along with their document file path. The document must be indexed prior to searching. When the user enters the query, it too will pass through all the preprocessing steps and will be stored inside a different index.

# Flow Chart Representation of major processes and code with explanation

**The Text pre-processing part has the following flow chart:**

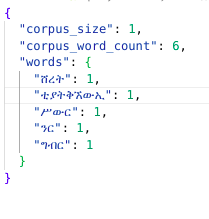


By doing these phases both information need and the documents can be represented and are made more easier for analysis since Guragigna is like every other language filled with ambiguity. The indexing processing is similar for the most part except in the definition of the indexing function it identifies if it is either a document or a query. Information retrieval (IR) systems are designed to help users find relevant information from a large collection of documents. In order to do this, IR systems must be able to represent both the information need (i.e., the user's query) and the documents in a way that makes them easy to analyze. This is particularly challenging when dealing with languages like Guragigna that are filled with ambiguity.  
 The indexing process is a key component of IR systems. During indexing, each document in the collection is analyzed and assigned a set of index terms that describe its content. These index terms are then used to create an inverted index, which allows the system to quickly locate documents that match a given query. In most cases, the indexing process is similar for both documents and queries. However, there are some differences in how the indexing function is defined for each. When indexing documents, the function must take into account the unique characteristics of each document and assign appropriate index terms based on its content. When indexing queries, on the other hand, the function must be able to identify the user's information need and generate appropriate index terms based on that need.

This is how the inverted index is stored, with the word as reference to the document which has the number next to it that tells the not normalized term frequency in that document.



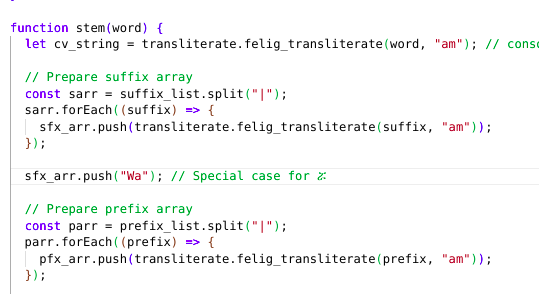
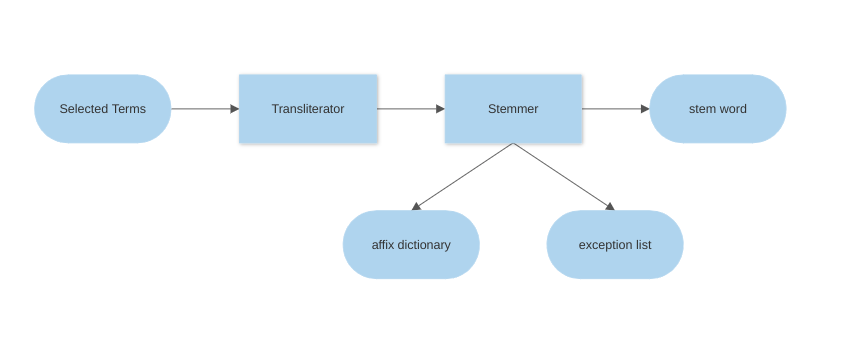
The index for the query carries less detail, it only contains number of words inside the query, the query terms and the term frequency(not normalized) of the term;

 These index files will be processed later on during the informational retrieval where there weights will be calculated using tf\*idf( tf here is normalized using corpus word count aka length)

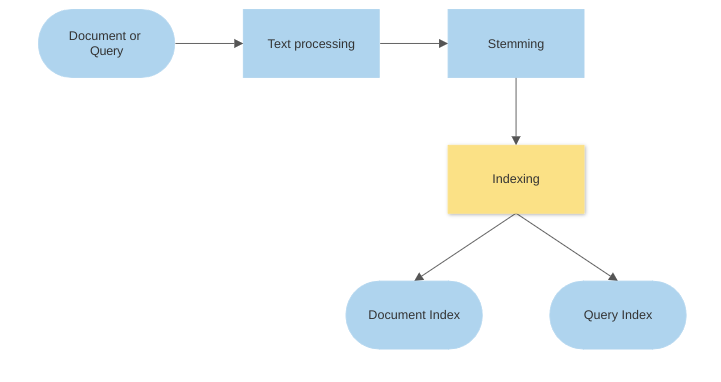
Lexical analysis and tokenization are called before stemming. Their task it to read the document and find words, by identifying first whitespace(“ “) then every thing separated by space is assumed to be a word then symbols, abbreviations and numbers are removed. Then it goes through a stoplist to remove common frequent words that don’t differentiate the documents from one another.

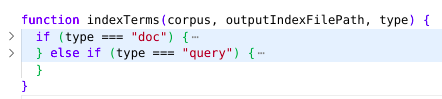
This function goes through a string array of stoplist words and removes them from the corpus prior to stemming process.

This is the flow chart of the stemming process. Prior to stemming the words to be indexed should already be selected.

This is a snippet code of the stemmer that transliterates the affixes(both suffixes and prefixes).

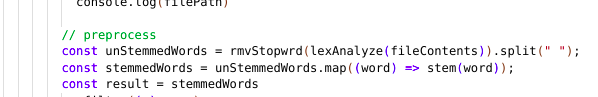
The stemmer also checks certain words in the exception list and if it finds them before or after iterative stemming, it will stop the stemming process and return the word.

 Indexing can happen on both documents and queries. This differentiation is made by a parameter that is passed to the indexer function.



The “Type” parameter decides how the input is indexed.

The “outputIndexFilePath” is a path index is stored.

Inside the indexer function, the document is imported, preprocessed and stemmed. Here is a snippet of code that shows this process.

There are a few assumptions during text processing in our project:

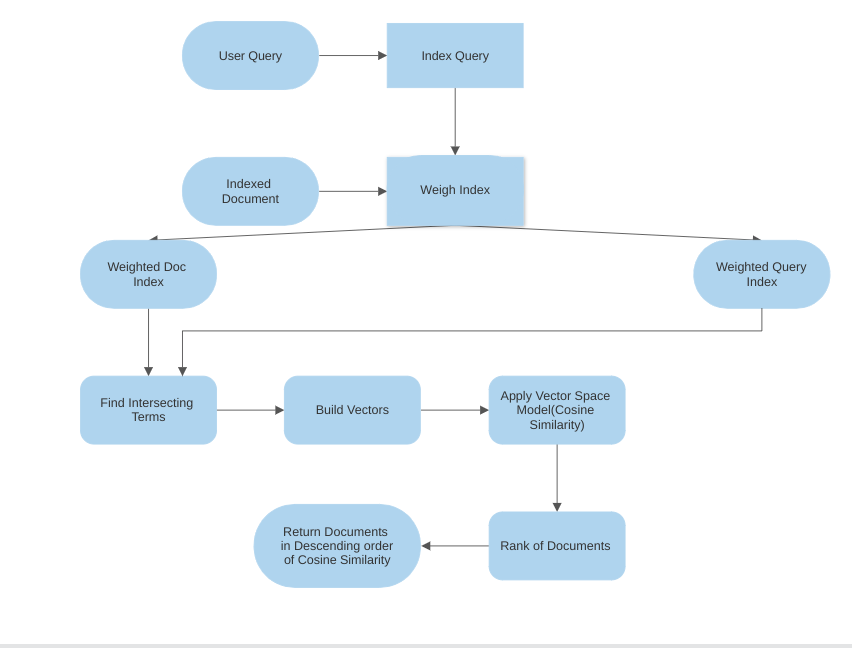
1) The document and query must be in Guragigna language.

2) Non-semetic words cant be stemmed.

3) The document must be a text file, binary files like docx cant be opened.

4) The query is presented to the search function in the form of a string.

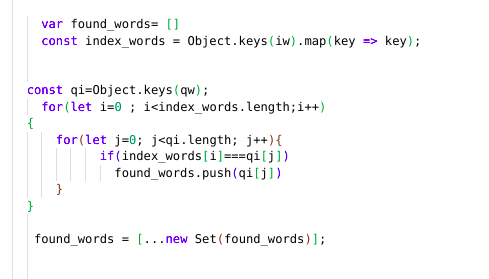
**The matching and retrieval part has the following flow chart:**



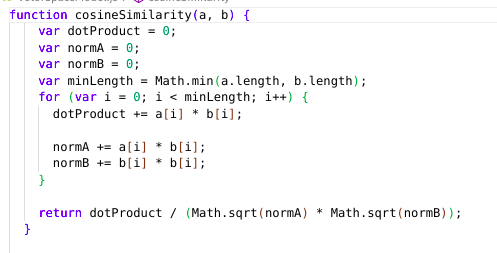
Information retrieval systems are designed to retrieve relevant information from a large collection of data. The process of retrieving information involves two main aspects: matching and retrieval.   
Matching involves comparing the query entered by the user with the stored index. This is be done using keyword searching. Retrieval involves finding all potential matches between the query and document and ranking them based on some measure of “similarity” so that the best matches receive the highest rank. The algorithms used for matching the query and index elements are based on a particular retrieval model. In this project we have used the vector space model, specifically cosine similarity to rank relevancy of documents.

This snippet of code from the search function shows the weighting of document index and the creation and weighing of query index.

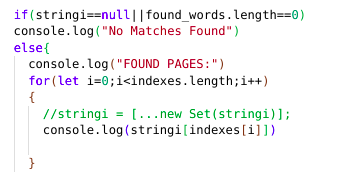
 This code is when the function tries to find intersecting terms between the query and the document:



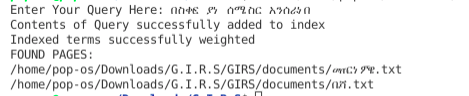
This function takes query vector “a” and document vector “b” to calculate their angle in the vector space.



The follow code snippet is what is finally returned to the user, the variable “stringi” is the ranked list of matching documents and the variable “found\_words” is the intersecting words between query and documents,



During the retrieval and matching aspect, our project assumes that the user isn't looking for specific words in a document and that the document is already indexed(inverted index).

This is the interface users interact with. It accepts string query and returns path of documents.

This concludes the description of our project, thank you!!